THALES

Gemalto Web Signer 1.8

INTEGRATION GUIDE

Document Information

Product Version	1.8
Document Number	D1571637A
Release Date	28 March 2022

Revision History

Revision	Date	Reason
Revision 1.1	13 December 2017	Initial release
Revision 1.2	26 March 2018	Additional sections: "Web Signer vs eSigner", "SWYS message and charset handling", "SWYS message length limits", and "Web Signer Invocation".
Revision 1.3	13 December 2018	New parameters added and new section
Revision 1.3.2	25 July 2019	Added appendix A "Comparison of POST messages between Web Signer and eSigner .
Revision 1.4	12 November 2019	Updated Data-certfilter_ku and unauthenticated attributes details.
Revision 1.4	28 January 2020	Added Logging chapter.
Revision 1.5	26 June 2020	Changed name of "data-cfg_signature_format" parameter to "data-signature_type
Revision 1.6	17 December 2020	Supported parameters table updated.
Revision 1.7	15 September 2021	More details about manual invocation. Firefox 50-52 notes removed. reply_tid parameter added.
Revision 1.7.1	12 January 2022	Added support for IdPrime MD 930 cards
Revision 1.8	28 March 2022	Added two new optional input parameters data-mime_ type and data-signature_param. Also added support to sign transactions with currency symbols.

CONTENTS

PREFACE: About This Document

This document describes how a web application can call Gemalto Web Signer through the used browser and provides the set of parameters that are supported in Gemalto Web Signer. It also addresses topics that will assist integrators in effectively developing Web server applications for use with Gemalto Web Signer.

It contains the following chapters:

This preface also includes the following information about this document:

For information regarding the document status and revision history, see "Document Information" on page 2.

Audience

The Gemalto Web Signer Integration Guide document is intended for programmers, developers, integrators and technical users who are familiar with developing Web applications

All products manufactured and distributed by Thales DIS France S.A. are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

It is assumed that the users of this document are proficient with security concepts.

Document Conventions

This document uses standard conventions for describing the user interface and for alerting you to important information.

Notes

Notes are used to alert you to important or helpful information. They use the following format:

> **NOTE** Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss. They use the following format:

> **CAUTION!** Exercise caution. Contains important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury. They use the following format:

> ****WARNING** Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.**

Command Syntax and Typeface Conventions

Format	Convention
bold	The bold attribute is used to indicate the following: > Command-line commands and options (Type **dir /p**.) > Button names (Click **Save As**.) > Check box and radio button names (Select the **Print Duplex** check box.) > Dialog box titles (On the **Protect Document** dialog box, click **Yes**.) > Field names (**User Name**: Enter the name of the user.) > Menu names (On the **File** menu, click **Save**.) (Click **Menu** > **Go To** > **Folders**.) > User input (In the Date box, type April 1.)
italics	In type, the italic attribute is used for emphasis or to indicate a related document. (See the *Installation Guide* for more information.)

Format	Convention
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[**optional**] [<optional>]	Represent optional **keywords** or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task.
{**a**|**b**|**c**} {<a>|<b>|<c>}	Represent required alternate **keywords** or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.
[**a**|**b**|**c**] [<a>|<b>|<c>]	Represent optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.

Related Documents

The following documents contain related or additional information:

> Web Signer Windows Release Notes

> Web Signer Mac Release Notes

Support Contacts

If you received Gemalto Web Signer from a distributor or a bank, as is typically the case, you should first contact the company who supplied the Gemalto Web Signer software to you. If you purchased Gemalto Web Signer directly from Thales, the support procedures should be described in your Support and Maintenance contract.

Further help is provided in the Self Support portal at https://supportportal.thalesgroup.com/.

Otherwise you can find information on how to contact your Thales representative in the Contact Us page at the Thales web site https://supportportal.thalesgroup.com/.

If You Find an Error

Thales makes every effort to prevent errors in its documentation. However, if you discover any errors or inaccuracies in this document, please inform your Thales representative. Please quote the Document Number found in "Document Information" on page 2.

CHAPTER 1: Overview

This chapter provides an overview of Gemalto Web Signer and contains the following topics:

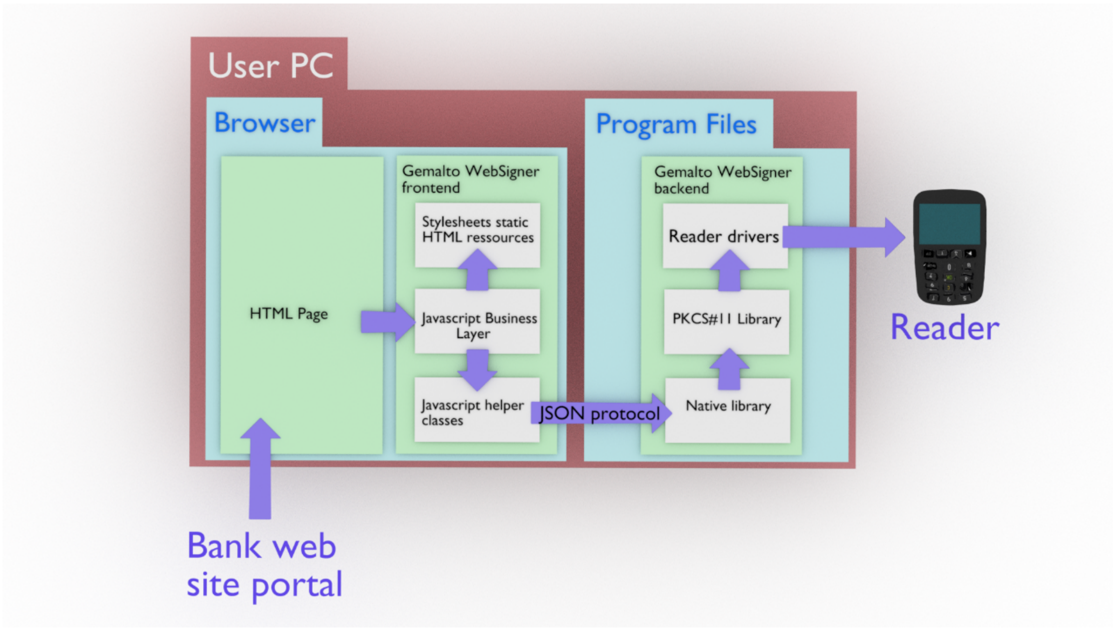
About Gemalto Web Signer

Gemalto Web Signer is a digital signature−based software solution designed to protect and secure Internet transactions. The Gemalto Web Signer solution is capable of signing documents or transaction data presented by a web application, verifying the authenticity of the signatories, and encrypting and decrypting documents. The verification of digital signatures has traditionally been the responsibility of the server. However, transactions have evolved from one-way exchanges of information to multiple exchanges going in both directions, from server to client to server. There is therefore a need for the same level of security on the client side as on the server side. Gemalto Web Signer provides this security. Gemalto Web Signer combines the privacy, integrity, and authentication functionality provided by cryptographic algorithms with simplicity, portability, and convenience. Your private key, the digital certificate, and other personal information are securely stored in a token, for example, a smart card, to prevent the fraudulent use of your electronic identity. With Gemalto Web Signer, you benefit from hardware-based security and the software-based security of PIN codes. Hardware-based security is a principal advantage that is significantly more secure than software-only solutions.

Gemalto Web Signer web extension

The Gemalto Web Signer solution consists of several components that work separately and can be packaged together as needed.

Figure 1: Gemalto Web Signer web extension components


User PC
Browser
HTML Page
Gemalto WebSigner frontend
Stylesheets static HTML ressources
Javascript Business Layer
Javascript helper classes
JSON protocol
Program Files
Gemalto WebSigner backend
Reader drivers
PKCS#11 Library
Native library
Reader
Bank web site portal

These components include:

- The Gemalto Web Signer web extension composed of various Javascript components
- The Gemalto Web Signer native libraries.
- Classic Client PKCS #11 library.
- Drivers for specific readers.

The Gemalto Web Signer digital signature web extension is launched by specific tags and embedded parameters in HTML pages loaded from the web application through a standard browser (Chrome, Firefox, Edge, Safari). These tags trigger a request to sign data that is either embedded in the HTML page or contained in a remote document, referenced by the page if the rendered area is too large.

The PKCS#11 cryptographic module

Gemalto Web Signer uses a PKCS #11 module to perform cryptographic functions. The PKCS #11 module, working together with Gemalto Web Signer, also manages access to cards for many purposes, including key generation, data encryption, and authentication. PKCS #11 refers to a standard that specifies an API, called Cryptoki, to devices which hold cryptographic information and perform cryptographic functions. Cryptoki, short for cryptographic token interface, follows a simple object-based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a cryptographic token. Gemalto Web Signer can use any PKCS#11 API. It is often used in conjunction with other Thales PKCS#11 libraries such as Classic Client.

Switching between Web Signer and eSigner

NOTE This section applies to Windows only as eSigner is no longer available for macOS.

Web Signer uses web extensions to sign web pages and verify these signatures. However, some old web browsers do not support web extensions and use an older format called NPAPI. Consequently Web Signer is designed so that if a page to be signed must be compatible with old web browsers, then eSigner comes into play instead.

The following table shows which browsers use Web Signer and which use eSigner:

Browser and Version	Web Signer	eSigner
Chrome	Y	
Edge	Y	
Internet Explorer		Y
Firefox > 52	Y	

For all the cases in the table, it is easy to make Web Signer adapt to the web browser.

To do this, put both `<div id="websigner"...>` and `<embed type="application/x-esigner"...>`. This means the appropriate product will be displayed according to the browser's capabilities (eSigner for Internet Explorer, Gemalto Web Signer for other browsers). Web Signer, if it is loaded in the page, will remove the eSigner `embed` element from the page as one of its first actions.

CHAPTER 2: Installation

This chapter describes the Installation and contains the following sections:

> "Setup" below

> "Browser store" on the next page

Setup

Publishing the web extension in the web store ensures that the content will be signed by the browser vendor. Some of the content needs to be republished if it is modified.

Use the Gemalto Web Signer setup file to install the following components:

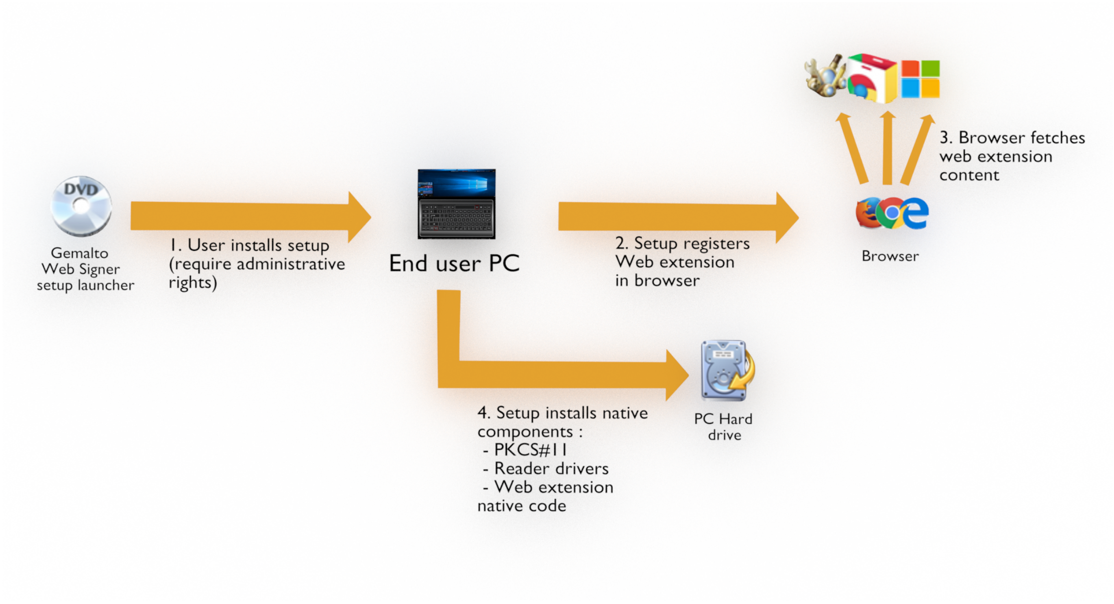
1. A native executable file
2. A Firefox web extension (via the Mozilla web store)
3. Chrome web extension (via the Chrome web store)
4. Edge web extension (via the Edge web store)

> **NOTE** The setup uninstalls previous versions of Gemalto Web Signer, eSigner and Classic Client cleanly, so you do not need to worry about uninstalling them manually.

The user will require administrative rights to install the above setup file.

The installer links the browser with the web extension automatically, however it requires user approval. In the case of Firefox, the installer asks you if want to install "an add-on" immediately after the installation wizard has finished. In the case of Chrome and Edge, this occurs the next time you start Chrome or Edge (you are asked if you want to activate a web extension).

Figure 2: Gemalto Web Signer setup


Gemalto Web Signer setup launcher
1. User installs setup (require administrative rights)
End user PC
2. Setup registers Web extension in browser
Browser
3. Browser fetches web extension content
4. Setup installs native components :
- PKCS#11
- Reader drivers
- Web extension native code
PC Hard drive

The setup can be launched in a terminal for silent installation by using the extra /silent parameter.

CAUTION! Disabling the command prompt script processing under the 'Prevent access to the command prompt' policy prevents Web Signer from working in Chromium-based browsers in Microsoft Windows.

The equivalent registry entry is:

```
[HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Windows\System]
"DisableCMD"=dword:00000001
```

Browser store

The web extension is stored in the browser store:

- Chrome https://chrome.google.com/webstore/category/extensions

- Edge https://microsoftedge.microsoft.com/addons/Microsoft-Edge-Extensions-Home

- Firefox Mozilla https://addons.mozilla.org/en-GB/firefox/

The Gemalto Web Signer web extension for the bank is NOT publicly available, but will be downloaded automatically from the adequate repository during the installation.

CHAPTER 3: Updates

This chapter discusses updates and contains the following topics:

> "Updating the Web extension" below

> "Updating native files" below

Updating the Web extension

Updates will be pushed automatically by the browser store, upon Customer approval.

Updating native files

The native files and backend web extensions can be updated at any time by the user by re-running the setup launcher.

CHAPTER 4: Update scenarios

This chapter describes various update scenarios as follows:

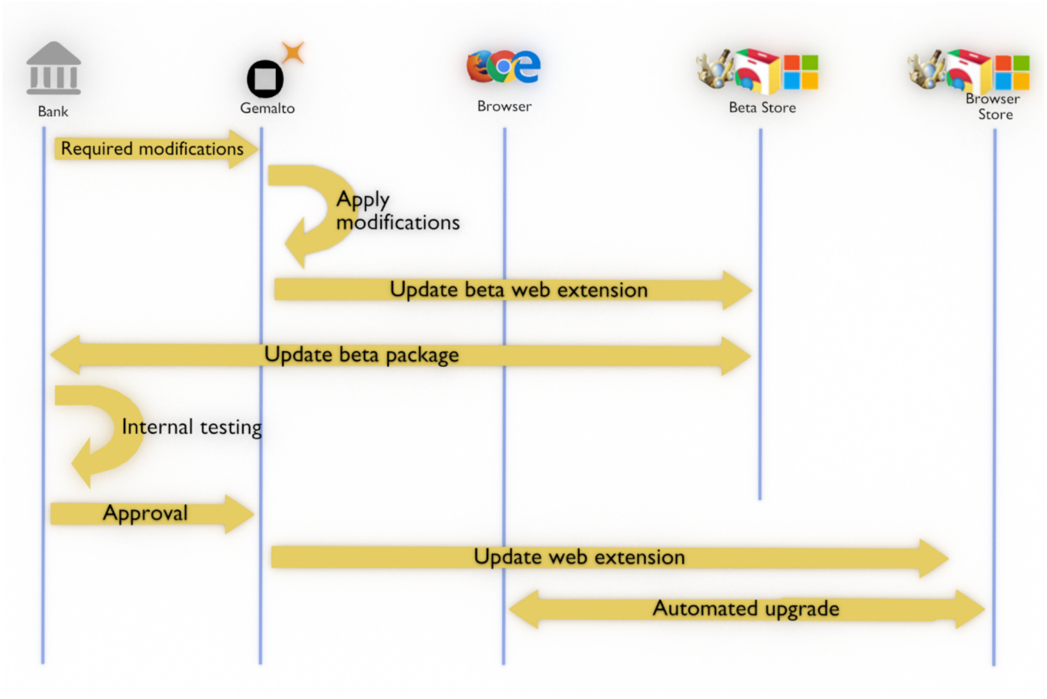
> "Scenario 1 – Web extension update only" below
> "Scenario 2 – Native files update only" on the next page
> "Scenario 3 – Major software update – Native + Web Extension " on page 18

Scenario 1 – Web extension update only

The Bank or Thales requests a change that requires only a web extension update - not the native files. For example, changing graphical elements, labels, links, and so on.

In this scenario, Thales prepares a test version of the web extension, the Bank approves it. Thales makes the new version available and the web extension is pushed to Customers via the Browser Store automatically.

Figure 3: Web extension update only

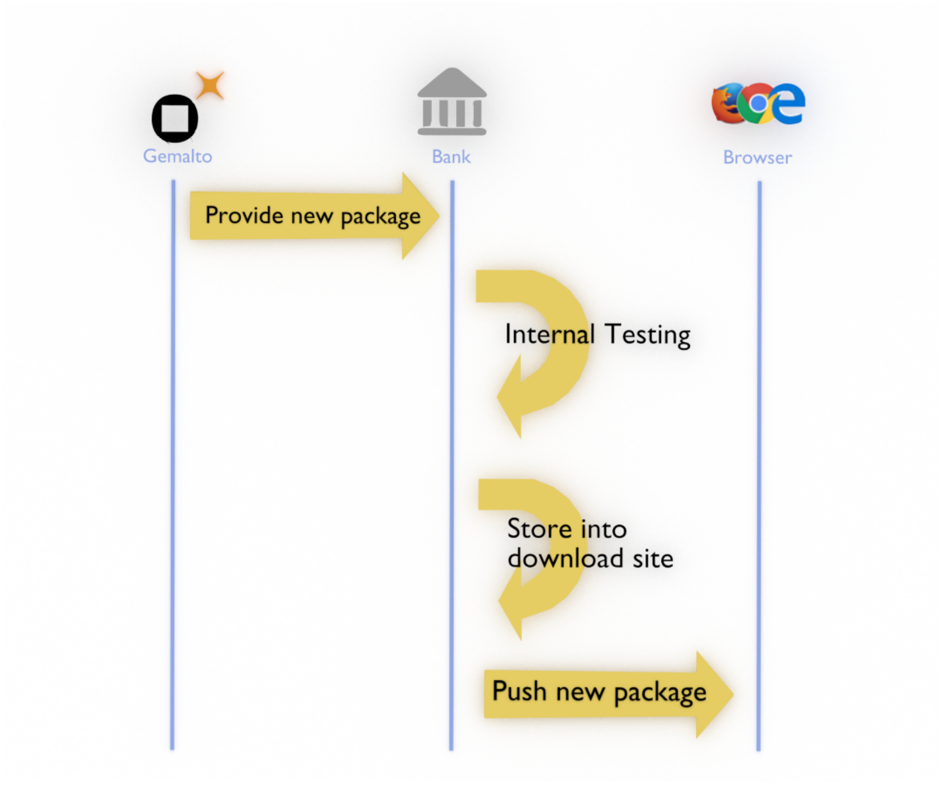


Scenario 2 – Native files update only

The Bank or Thales requests the modification of native files, which are not mandatory for software usage. For example, minor text changes, updates, and so on.

In this scenario, Thales prepares the .msi test version of the web extension, the Bank approves it. Thales makes the new version of the setup package available on the download site, and the Bank can propagate this package to the new Customers.

Figure 4: Native files update only

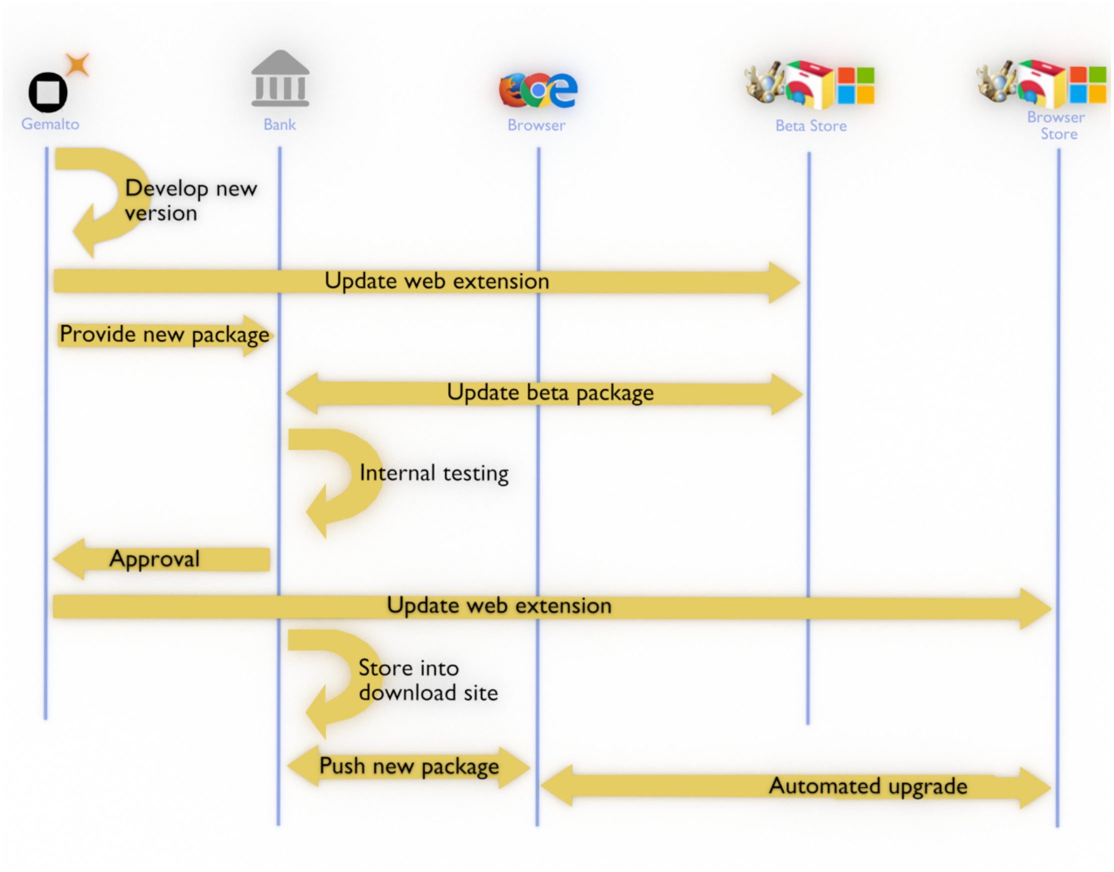


Scenario 3 – Major software update – Native + Web Extension

The Bank or Thales requests the modification of Gemalto Web Signer, in order to add a new feature, fix issues, or security update.

In this scenario, Thales prepares the .msi test version and a new version of the web extension, the Bank approves it. Thales makes the new version of the setup package available on the download site, and a test version in the Browser Store test account. Once approved, the Bank can propagate this package to the new Customers. The Web Extension is automatically updated via the Browser store. The Web Extension will forward the user to the update procedure until the native layer has been upgraded.

Figure 5: Native files and web extension update



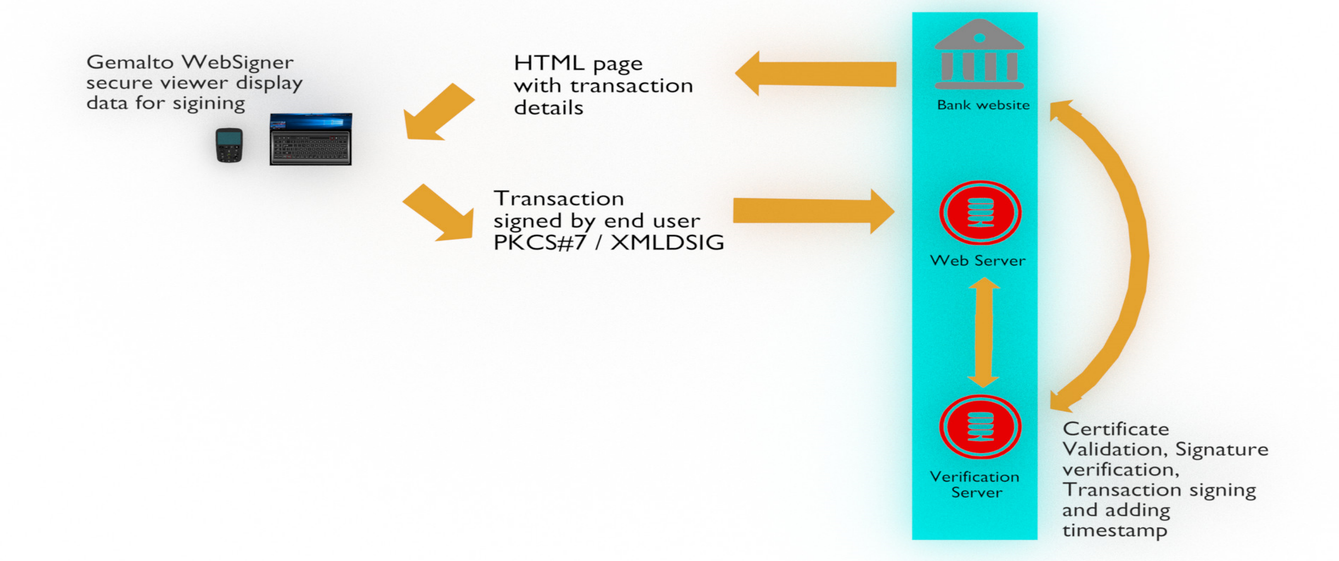
CHAPTER 5: Signature and verification usage

This chapter describes the input data. It contains the following topics:

Signature and verification components

During the signature the following components are used:

Figure 6: Gemalto Web Signer components for signature and verification


Gemalto WebSigner secure viewer display data for sigining
HTML page with transaction details
Transaction signed by end user PKCS#7 / XMLDSIG
Bank website
Web Server
Verification Server
Certificate Validation, Signature verification, Transaction signing and adding timestamp

Bank website

The bank website provides the HTML code to invoke the web extension and provide data content to sign. Please see "Examples" on page 50 for details of the HTML code.

User PC

The Web Extension parses the page, to look for Gemalto Web SignerHTML code. Once found, it triggers the appropriate call to the Native layer.

Native layer

Provides binary components to communicate with the middleware driver layer and smart card for signature. The smart card signature is processed and formatted into a compatible format, such as PKCS#7 and forwarded back to the Web Extension for POSTing.

Bank verification server

The Bank verification is a web service that validates the POSTed signature for authenticity and integrity. It is up to the Bank to choose the best verification mechanisms.

Web Signer invocation

The Web Signer GUI can be displayed automatically or manually, according to the presence of a "websigner div" (<div id="websigner"...> element) when the page is loaded.

Whenever a page is loaded and its document object model (DOM) is complete the browser checks if the page's URL passes the Web Signer page whitelist (specified in Web Signer's manifest.json file). If the page passes this filter, the Web Signer extension code runs on it. The following events happen:

1. Web Signer inserts a `<meta>` element in the page's head section with the following content: `<meta name="WebSigner" content="is installed">`
2. A custom event handler for events named 'webSignerParse' is installed.
3. The DOM is checked for the presence of `<div id="websigner"...>` elements. Depending on the number of such elements found an action is taken.
   a. If there is no such element, nothing happens. See section ""Manual invocation" below" for details.
   b. If there is exactly one `<div id="websigner"...>`element, the DOM is checked for any other elements that might interfere with the Web Signer's GUI HTML. If no offending elements are found the Web Signer GUI is injected, otherwise an error message is displayed - "GUI_PARTS_ALREADY_PRESENT". Refer to "Offending elements" on the next page for a list of elements that might interfere with the Web Signer GUI.
   c. If there is more than one`<div id="websigner"...>`element, an error message displays (injected in place of Web Signer's GUI) - "TOO_MANY_WEBSIGNER_DIVS".

Manual invocation

If a page is displayed by default without any websigner div and some prior user action is needed, the websigner div element will be inserted into the page's DOM. Once websigner div is present in the page's DOM you can fire an event webSignerParse manually and processing will continue from point 3 again.

One exception to this is that in step 3a, an error message displays if there are no `<div id="websigner"...>` elements in the DOM ("NO_WEBSIGNER_DIV"). The purpose of the 'webSignerParse' event is to start the Web Signer when the original page did not contain the `<div id="websigner"...>` element and it was inserted later on.

How to fire a webSignerParse event from page

When you want to manually control displaying Web Signer, first add the necessary content and then fire a "webSignerParse" event like a calling method in the example.

```
<script>
        function FireWebSignerParseEvent() {
                var myEvent = new Event('webSignerParse');
                document.dispatchEvent(myEvent);
        }
</script>
```

Offending elements

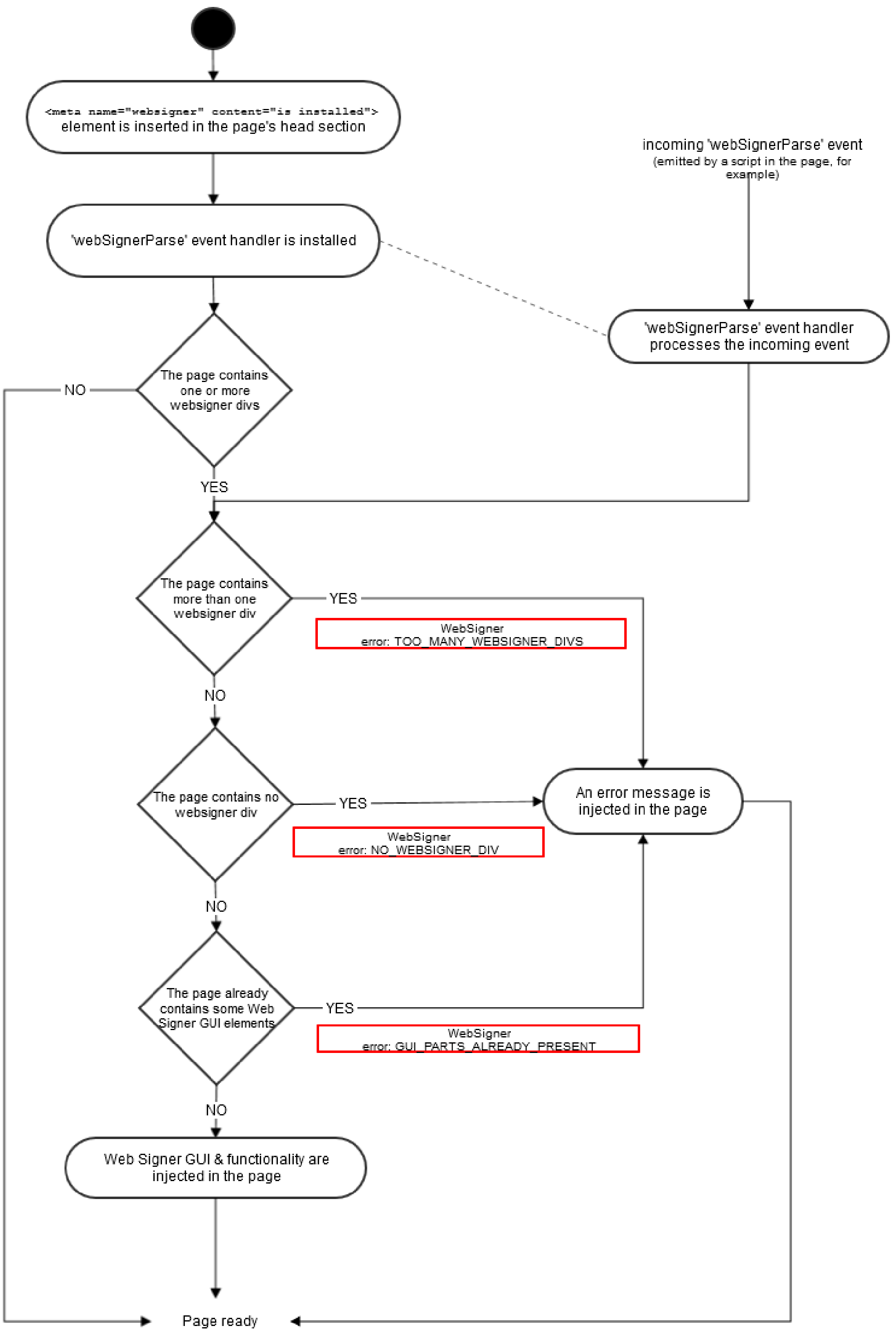
The following table shows the elements mentioned above in 3b that are parts of the Web Signer GUI, that are injected by Web Signer and need to be removed before trying to invoke Web Signer.

Name & Tag	Description of GUI part	How to select for removal
SuperWrapper <div>	Web Signer's GUI itself	These elements all share the "data-websigner" attribute for easier removal. This means for example `document.querySelectorAll('[data-websigner]')` will select them.
toBePrinted <div>	Web Signer's container for printing	
Web Signer <div>	Where the parameters for Web Signer are set	`<div id="websigner" ... >` can easily be selected via `document.getElementById('websigner')`

You can either remove these elements manually, or use the new event called webSignerRemove (introduced in Web Signer 1.3). Sending this event will remove all the Web Signer GUI elements in the page, although the original <div id="websigner" ... > will be kept.

The following figure summarizes the process:

Figure 7: Web Signer invocation process



<meta name="websigner" content="is installed">
element is inserted in the page's head section
'webSignerParse' event handler is installed
incoming 'webSignerParse' event
(emitted by a script in the page, for example)
'webSignerParse' event handler processes the incoming event
The page contains one or more websigner divs
NO
YES
The page contains more than one websigner div
YES
WebSigner error: TOO_MANY_WEBSIGNER_DIVS
NO
The page contains no websigner div
YES
WebSigner error: NO_WEBSIGNER_DIV
An error message is injected in the page
NO
The page already contains some Web Signer GUI elements
YES
WebSigner error: GUI_PARTS_ALREADY_PRESENT
NO
Web Signer GUI & functionality are injected in the page
Page ready

The Manifest file

Purpose of the Manifest file

The Manifest file, manifest.json, contains information relating to the context of the Web Extension. For example, we can find the different URL addresses the web extension can communicate with. If this value is not set with the right address, the Web Extension will probably fail to execute.

For more information visit one of the following:

For Firefox: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json

For Chrome: https://developer.chrome.com/extensions/manifest.

For Edge: https://docs.microsoft.com/en-us/microsoft-edge/extensions-chromium/getting-started/manifest-format

Modifying resources

It is possible to change images and media files. The assets directory contains the different images and css that are downloaded when the user downloads the web extension. Those files are signed and cannot be changed by anyone once they are downloaded on the User PC.

CHAPTER 6: Input data

This chapter describes the input data. It contains the following topics:

Supported parameters

Parameter	Description	Value	Comment
data-mode	Working mode of the extension.	"sign", "button", “verify”	Optional. > "sign" - extension runs in signing mode. > "button" - extension runs in button mode. > “verify” – extension runs in verification mode. Default value is "sign" (if the attribute is omitted or is not of a valid value). **NOTE** The values must be written entirely in lowercase.
data-button_label	Label of the sign button in Button mode	String	Optional. Default value is "Sign".
data-tdw_height	Height (in pixels) of the window that displays the content to be signed.	Positive integer	Optional. Default value of text data window is 260 (pixels). Overridden by gui_height parameter.
data-tdw_width	Width (in pixels) of the window that displays the content to be signed.	Positive integer	Optional. Default value of text data window is 600 (pixels). Overridden by gui_width parameter.

Parameter	Description	Value	Comment
data-gui_height	Height (in pixels) of Gemalto Web Signer GUI.	Positive integer	This optional parameter supersedes the tdw_height parameter. If gui_height value is undefined or incorrect, GUI size uses tdw_ height value. If, in turn, the tdw_height parameter is undefined or incorrect, tdw_height default value is used.
data-gui_width	Width (in pixels) of Gemalto Web Signer GUI.	Positive integer	This optional parameter supersedes the tdw_width parameter. If gui_width value is undefined or incorrect, GUI size uses tdw_ width value. If, in turn, the tdw_width parameter is undefined or incorrect, tdw_width default value is used.
data-page_height	Height (in pixels) of a single page displayed in the window.	Positive integer	Optional. Default value is 260 (pixels). If the height of the page is larger than the data window height a vertical scroll bar is displayed. If the data to be signed does not fit a single page, multi-page controls are shown allowing the user to browse the data page by page.
data-in_data	Data to be signed.	String	Either this or data-in_data_url parameter is required. In case of “verify” mode, signature data must be encoded in base64.
data-in_data_encoding	Specifies encoding of the data to be signed.	String	Optional. Applies to data specified through the data-in_data attribute only. > If set to "url" - Web Signer decodes url encoded characters. e.g. %20 is decoded as a space. The "+" symbol is also decoded as a space. > If set to any other value, Web Signer does not decode url-encoded characters.
data-reply_url	URL that shall receive the signature	String	Mandatory. When the signature is complete, the binary data of the signature will be POSTed to this URL along with additional information.

Parameter	Description	Value	Comment
data-reply_target	Specifies a frame name or a keyword that indicates where to POST results	 _blank _self _parent _top *framename*	Optional. Displayed in a new window or tab . Displayed in the same frame (default). Displayed in the parent frame. Displayed in the full body of the window. Displayed in a named frame. According to the standard HTML TARGET attribute.
data-in_data_url	Provides the URL from which the data to be signed is downloaded.	String	Either this or data-in_data parameter is required.
data-certfilter_ku	Specifies the filter applied to the certificate's key usage extension	String	Optional. Specifies the filter to apply to the certificate's keyUsage extension. For coding details see "Data-certfilter_ku details" on page 29 Default (if not specified) is 128:0
data-mime_type	Specifies if the document to be signed is in HTML or plaintext format.	String	Optional. Possible values : > text/plain (for plaintext document) (default) > text/html (for HTML document)
data-signature_type	Describes the signature format that is to be generated.	String	Optional. Possible values : > pkcs7_basic (default) > pkcs7_detached Not yet supported: > xmldsig

Parameter	Description	Value	Comment
data-signature_param	Name of POST parameter containing the base64 and urlencoded PKCS#7 signature	String	Optional. Possible values : > Signature_Data (default) > Signature > Signature_Data, Signature (order does not matter) Please note that if both Signature_Data and Signature are defined (separated by a comma), Web Signer creates two POST parameters containing the same base64 and urlencoded PKCS#7 signature. It is important to highlight that while the HTTP POST method does not have any data size limits, different HTTP servers and browsers do have limits regarding the HTTP POST data size. Please consult the specific documentation for your HTTP server and the browsers you support with your application, in which Web Signer is embedded.
data-cfg_sign_hash	Specifies the Hash algo to use. SHA256 as default	String	Optional. SHA256 is the only supported algorithm at present.
data-swys_message	Parameter used for starting a transaction using SWYS compatible Device.	String	Mandatory for SWYS signatures. Contains the text to be signed. If used, data-swys_ type must also be set. Maximum length is 200 characters, including control characters. The combined length of data-swys_ message and data-swys_hidden_data must not exceed 217 bytes.
data-swys_type	Specifies the type of SWYS signature	String	Mandatory if data-swys_message is specified. Possible value is: > oath_ocra_otp Value must be in lowercase.
data-swys_hidden_data	Contains the hidden data that should be part of the SWYS signature.	String	For SWYS signatures only. Optional. Maximum length is 127 bytes. The combined length of data-swys_ message and data-swys_hidden_data must not exceed 217 bytes.

Parameter	Description	Value	Comment
data-swys_reader_ fallback	Defines level of security (alternative reader that can be used if user is not using a SWYS reader.	String. Default is "none"	For SWYS signatures only. Optional. Possible values are: > none (reader must be SWYS) > pinpad (pinpad reader acceptable - but signature will be non-SWYS). Values must be in lowercase.
data-reply_tid	Transaction ID.	String	Optional. Is returned unchanged to server in the reply_tid POST parameter. Can be used to identify a session on the server.

Data-certfilter_ku details

The naming of the bits comes from RFC5280, section 4.2.1.3 (https://tools.ietf.org/html/rfc5280#section-4.2.1.3).

The following table shows the meaning of each bit in data-certfilter_ku.

b8	b7	b6	b5	b4	b3	b2	b1	Decimal	Meaning
1								128	Digital Signature
	1							64	Non-Repudiation
		1						32	Key Encipherment
			1					16	Data Encipherment
				1				8	Key Agreement
					1			4	Key Cert Sign
						1		2	Certificate Revocation List (CRL) Sign
							1	1	Encipher only

192 : Identity certificate

184: Utility certificate

Instead of “192” and “184”, the keywords “identity” and “utility” respectively may be used.

Single Value

The value of data-certfilter_ku is the sum in decimal of the bits set. For example, if the certificate can be used for Digital Signature and Non-Repudiation, the value is 128 + 64 = 192.

Two Values

Alternatively, data-certfilter_ku can also specify the bits that MUST be set to 0. In this case, the format is A:B where A codes the values that must be 1 and B codes the values that must be 0.

For example, a value of 192:8 means that bits 8 and 7 must be 1 and bit 4 must be 0. The other bits can take any value.

Here are some more examples:

64:128 - Non repudiation must be set (b7=1); digital signature must not be set (b8=0)

0:0 - can be any certificate

64:0 - Non-repudiation must be set (b7=1), the other bits can take any value.

192:0 - is the same as the single value 192 (b8 and b7 = 1, other bits can be any value)

Limitations

Filtering by the "Decipher Only" bit mentioned in RFC5280 is not supported by Web Signer due to backward compatibility reasons.

Specifying values of A above 255 yields no certificates, specifying values of B above 255 is equal to using the same value modulo'd 256 - only its lower 8 bits are taken into account.

Single certificate available for signature

If after applying the key usage filter there is only one certificate in the card that can be used for signatures, it is automatically selected for signing after the user clicks the **Sign** button. The name of the certificate selected is visible above the PIN entry message.

SWYS message encoding and character set handling

The data-SWYS_message attribute can contain special characters (ASCII 0x00 - 0x1F) and Unicode.

The backend binary maps these into the default SWYS reader character set ISO8859-15 with these possible outcomes:

> One or more characters cannot be mapped to the ISO8859-15 charset - an error is returned without sending the SWYS message to the reader.

> The SWYS message contains a NULL character (U+0000) - an error is returned without sending the SWYS message to the reader.

> The SWYS message contains a character that is not supported by the reader (generally, from the range of control characters only U+000A is accepted) - the SWYS message is sent to the reader but the reader returns an error. Refer to the reader's specification for a table of supported characters.

> All of the characters in the message are supported - the reader asks the user for PIN and displays the SWYS message.

Currently, the message is always converted into ISO8859-15 even though the reader might support other character sets (typically ISO8859-5). Selecting a different character set is not supported.

SWYS message length limits

SWYS_messages have the following length limits:

> The SWYS message (data-SWYS_message attribute) must not exceed 200 bytes in length.

> The hidden SWYS message (data-SWYS_hidden_data attribute) must not exceed 127 bytes in length.

> The combined length of SWYS message and Hidden SWYS message must not exceed 217 bytes.

CHAPTER 7: Output data

Currently, Gemalto Web Signer POSTs the following data to the server specified in reply url (data-reply_url parameter) upon successful signature or canceled-by-user action:

> "Successful signature" below

> "PKCS#7 structure" on the next page

> "Error messages" on page 36

> "Signing process cancellation" on page 40

> "Notes on the GUI" on page 40

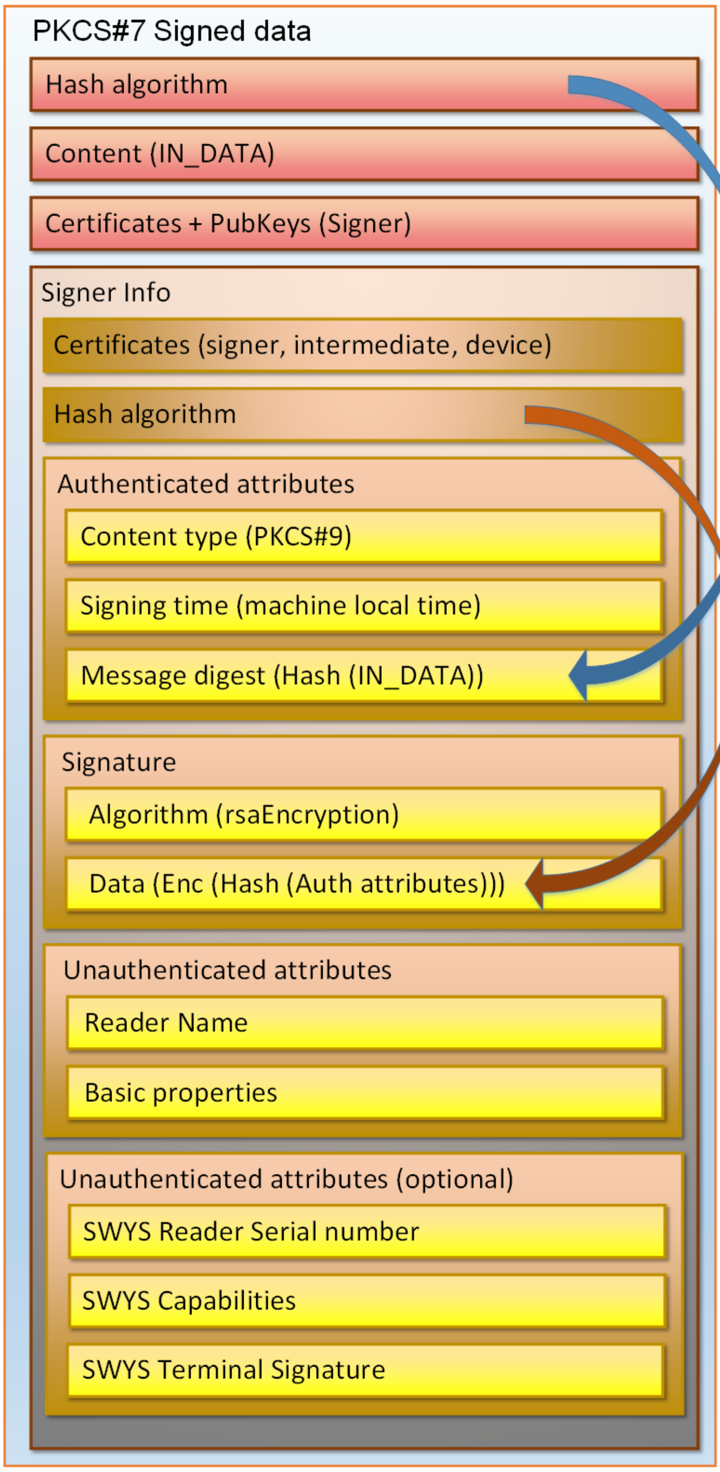
Successful signature

Parameter	Content	Example
Signature	PKCS#7 signature binary data encoded in base64.	
FrontendVersion	Official version of the frontend part of Gemalto Web Signer that has been used to generate the signature.	"0.0.17"
FrontendRevision	Internal code version identification (revision number).	"242"
FrontedRevHash	Internal code version identification (revision hash).	"a1e78cf87c58"
BackEndVersion	Version of the backend part of Gemalto Web Signer that has been used to generate the signature.	“1.0.0.1”

PKCS#7 structure

Gemalto Web Signer provides PKCS#7 data with the following structure:

Figure 8: PKCS#7 data structure


PKCS#7 Signed data
Hash algorithm
Content (IN_DATA)
Certificates + PubKeys (Signer)
Signer Info
Certificates (signer, intermediate, device)
Hash algorithm
Authenticated attributes
Content type (PKCS#9)
Signing time (machine local time)
Message digest (Hash (IN_DATA))
Signature
Algorithm (rsaEncryption)
Data (Enc (Hash (Auth attributes)))
Unauthenticated attributes
Reader Name
Basic properties
Unauthenticated attributes (optional)
SWYS Reader Serial number
SWYS Capabilities
SWYS Terminal Signature

NOTE Gemalto Web Signer includes only Signer certificates in the PKCS#7.

Unauthenticated attributes

These attributes concern the reader that was used to perform the signature. The reader name and basic properties are always present. The others are present only if the reader supports SWYS.

The reader attributes follow the ASN.1 defintions in RFC 3852 Cryptographic Message Syntax.

```
UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

        Attribute ::= SEQUENCE {

          attrType OBJECT IDENTIFIER,

          attrValues SET OF AttributeValue }

        AttributeValue ::= ANY
```

The following table provides a more detailed look at the individual attributes:

Table 1: Individual Unauthenticated Attributes Descriptions

Attribute name	attrType (OID)	AttributeValue	Attribute meaning
Reader name	1.3.6.1.4.1.31746.1100.10.4.2.2	UTF8String	Reader name as returned by the PKCS#11 module.
Reader serial number	1.3.6.1.4.1.31746.1100.10.4.2.1	UTF8String	Reader serial number as returned by the PKCS#11 module.
Basic reader properties	1.3.6.1.4.1.31746.1100.10.4.2.4	PrintableString	Indicates whether it is a hardware reader (HW_SLOT) or not (NOT_HW_SLOT) and whether it has protected authentication path (PINPAD).
SWYS reader capabilities	1.3.6.1.4.1.31746.1100.10.4.2.5	PrintableString	Indicates the SWYS modes available to the reader (OATH_OCRA_OTP, PKI_SIGNATURE) and possible digest mechanisms for SWYS (SHA256, SHA512).

Attribute name	attrType (OID)	AttributeValue	Attribute meaning
SWYS Terminal Signature	> 1.3.6.1.4.1.31746.1100.10.4.1.1 > 1.3.6.1.4.1.31746.1100.10.4.1.2	OctetString	> OATH OCRA OTP based SWYS signature > PKI based SWYS signature
Browser version	1.3.6.1.4.1.31746.1100.10.4.3.2	PrintableString	Version number of the browser in which the signature was created (major version number only).
Browser name	1.3.6.1.4.1.31746.1100.10.4.3.1	PrintableString	Name of the browser in which the signature was created.
SWYS message encoding	1.3.6.1.4.1.31746.1100.10.4.4.3	PrintableString	Encoding of the SWYS message.
SWYS hidden data	1.3.6.1.4.1.31746.1100.10.4.4.2	OctetString	Hidden data used in the SWYS signature.
SWYS message	1.3.6.1.4.1.31746.1100.10.4.4.1	OctetString	SWYS message used in the SWYS signature.

Example of unauthenticated attributes

```
unsignedAttrs:
    object: undefined (1.3.6.1.4.1.31746.1100.10.4.3.2)
    value.set:
      PRINTABLESTRING:68

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.3.1)
    value.set:
      PRINTABLESTRING:Firefox

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.4.3)
    value.set:
      PRINTABLESTRING:ISO-8859-15

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.2.1)
    value.set:
      UTF8STRING:K1650143760364

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.2.4)
    value.set:
      PRINTABLESTRING:HW_SLOT,PINPAD

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.4.2)
    value.set:
      OCTET STRING:
        0000 - 53 6f 6d 65 74 68 69 6e-67 20 68 69 64   Something hid
        000d - 64 65 6e                                 den

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.2.5)
    value.set:
      PRINTABLESTRING:OATH_OCRA_OTP,SHA256

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.4.1)
    value.set:
      OCTET STRING:
        0000 - 53 6f 6d 65 74 68 69 6e-67 20 74 6f 20   Something to
        000d - 62 65 20 64 69 73 70 6c-61 79 65 64      be displayed

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.2.2)
    value.set:
      UTF8STRING:Gemalto Ezio Shield Pro SC 0

    object: undefined (1.3.6.1.4.1.31746.1100.10.4.1.1)
    value.set:
      OCTET STRING:
        0000 - a1 29 c1 04 96 69 10 78-c2 02 00 ab e3   .)...i.x.....
        000d - 00 e5 01 0f e0 00 e1 00-e8 00 e9 04 a6   .............
        001a - c1 5b 6e ea 0c eb 01 04-ec 02 01 00 ed   .[n..........
        0027 - 03 2a 9e 9a                              .*..
```

Error messages

If errors are encountered during the processing of any transaction, an error message is returned. Generally, error messages are sent to the server which displays them for the user. However, as indicated below, some are displayed in the viewer on the user's computer instead of being sent to the server. The following table lists the error messages, their possible causes, and solutions.

The messages are being re-written in a more user-friendly style. Not all have yet been converted to this style. These "unconverted" ones appear at the beginning of this table.

Error message	Cause	Comment
DATA_NOT_ RETRIEVED	There was a problem retrieving the data to be signed, as defined by IN_ DATA_URL/DataURL. One of the following may have occurred - the specified URL was incorrect or the data could not be retrieved from the URL.	Not yet supported
INSUFFICIENT_ PARAMETERS	One or more mandatory parameters is missing.	Not yet supported
INVALID_ SIGNATURE_ FORMAT	This error occurs if the format of the signature given to Gemalto Web Signer to display is not supported.	Not yet supported
MIME_TYPE_NOT_ SUPPORTED	The specified MIME type is not one of the ones supported.	Not yet supported
OTHER_ERROR	An unspecified error occurred. This error should be returned only if an error occurs that cannot accurately be reported using one of the other error codes. The implementation may return a more detailed reason for the error in the message string of the exception.	Not yet supported

Error message	Cause	Comment
URL_INVALID	The URL defined by the data-in_data_url parameter either does not exist or is not a valid URL, e.g. www.google.fr (http://www.google.fr is valid because it specifies the protocol)..	Supported
USER_ABORT	The user has aborted the procedure.	Error sent to the server
HASHLIST_NOT_ MATCHED	The hash mechanism supported by the smart card does not match the list supported by the server.	Not yet supported
INVALID_READER	Reader cannot be used for signing transaction. Example, SWYS transaction.	Not yet supported
Internal error	A generic error that by default replaces all exceptions that have not yet been converted to user-friendly style. For example having the Hidden SWYS message longer than 127 bytes generates this error.	Supported
Login: PIN entry canceled	Two possible causes: > the PIN entry times out > the user holds the reader's Cancel button for 3 seconds while there are no PIN digits entered.	Supported. Pulling the card out of the reader generates a "Login failed" message. N.B. the time-out or cancel does not apply for PIN entry with a PINPad-less reader (CT30 or similar), since in this case the middleware Login function is called after the PIN is entered in Web Signer. For PINPads the PIN entry is part of the middleware Login function and is handled by the middleware, Web Signer just displays an informative message.
Login failed	The user pulls the card out from the reader during PIN entry. Also the default message for failure of the Login function in middleware when there is no specific error message available.	Supported

Error message	Cause	Comment
Login failed: Incorrect PIN given, remaining tries left: {N}	The user enters a wrong PIN. {N} is replaced by the number of PIN entry tries remaining. If Web Signer cannot retrieve the number of tries remaining, only the message "Login failed: Incorrect PIN given" is displayed.	Supported.
Login failed: PIN locked	The user enters a wrong PIN when there was only one PIN entry attempt left or the PIN is already (b)locked.	Supported
Sign process aborted. The certificate can't be used for signing: user PIN is not initialized.	The user is trying to sign a transaction using a card whose user PIN is not initialized.	The card's user PIN has to be initialized before it can be used to sign transactions.
The following messages are shown when the input HTML for Web Signer is erroneous and normally should not be seen by the end users. They are intended for integrators' eyes only.		
Swys message too long	The SWYS message is longer than 200 characters, including control characters.	Supported. N.B. for SWYS messages the number of characters does not neccessarily equal the number of bytes. In the source HTML, when using UTF-8, the special symbols are typically multibyte sequences but are converted to a ISO8859 charset to be displayed on a reader. In the ISO8859 charset, one character = one byte. Thus, the original data=swys_message attribute can be longer than the actual message given by the backend to the reader where the check for 200 character length is made.
INSUFFICIENT_ PARAMETERS: swysType	The **data-swys_type** attribute is missing in the `<div id="websigner">` element. This attribute is mandatory when the attribute **data-swys_message** is specified.	Supported

Error message	Cause	Comment
INSUFFICIENT_ PARAMETERS: swysMessage	The **data-swys_message** attribute is missing in the `<div id="websigner">` element. This attribute is mandatory when the attribute **data-swys_type** is specified.	Supported
WRONG_ PARAMETER: swysType='{value}	The **data-swys_type** attribute in the `<div id="websigner">` element contains a value that is not one of the supported SWYS type ("oath_ocra_otp" or "pki_ signature"). The {value} is replaced with the attribute's value.	Supported

Signing process cancellation

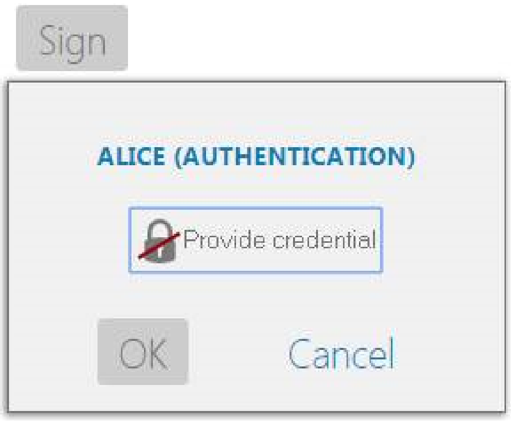
This could invoked by User clicking on the Cancel button, or cancelling signature process through the reader, timeout.

Parameter	Content	Example
Exception	"USER_ABORT"	"USER_ABORT"
FrontendVersion	Official version of the frontend part of Gemalto Web Signer that has been used to generate the signature.	"0.0.17"
FrontendRevision	Internal code version identification (revision number).	"242"
FrontedRevHash	Internal code version identification (revision hash).	"a1e78cf87c58"
BackEndVersion	Version of the backend part of Gemalto Web Signer that has been used to generate the signature.	“1.0.0.1”

Notes on the GUI

Padlock icon for PIN entry

When using a non-PIN Pad reader, Web Signer displays a modal window for the user to enter a PIN. The padlock icon on the left of the PIN input element tries to alert the user of security risks. When the page where the transaction is being signed is not using the secure protocol (HTTPS), the padlock icon appears crossed by a red line.



****WARNING**** **The padlock icon should be regarded as an indication of security risk but NOT be taken to be a guarantee of security on its own. For example, it will be displayed as a secure padlock when using the HTTPS protocol even if the server certificates are considered not secure enough by the browser.**

CHAPTER 8: Logging

This chapter describes logging and contains the following topics:

The Web Signer extension provides logging facilities to collect information for the purposes of troubleshooting potential problems of the extension. Sensitive data (transaction data, PINs, and so on.) are not stored within the logs. The controls to start and to stop logging are located in the extension options page. The following section explains how to access the controls.

> **NOTE** The log file folder has to exist for the Gemalto Web Signer to write the logs. Refer to the Log folders section below for more information.

Accessing extension options

This section describes how to access the extension's options page, for each browser.

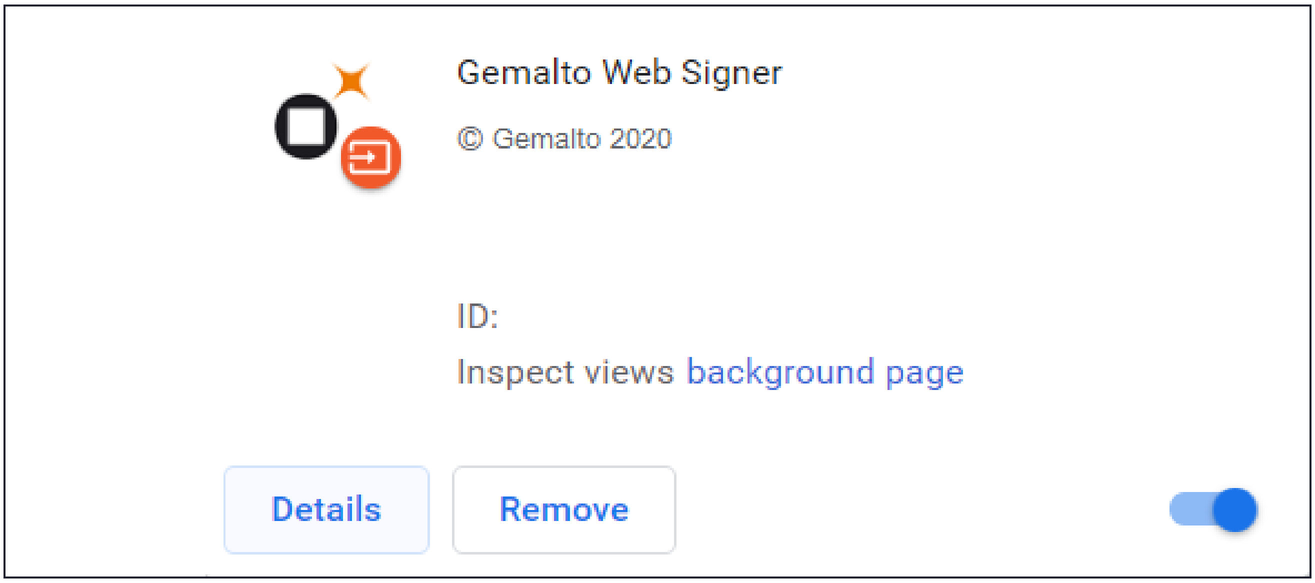
Chrome and Edge

The method is the same for both browsers, but the screens may look slightly different. The screenshots show Chrome.

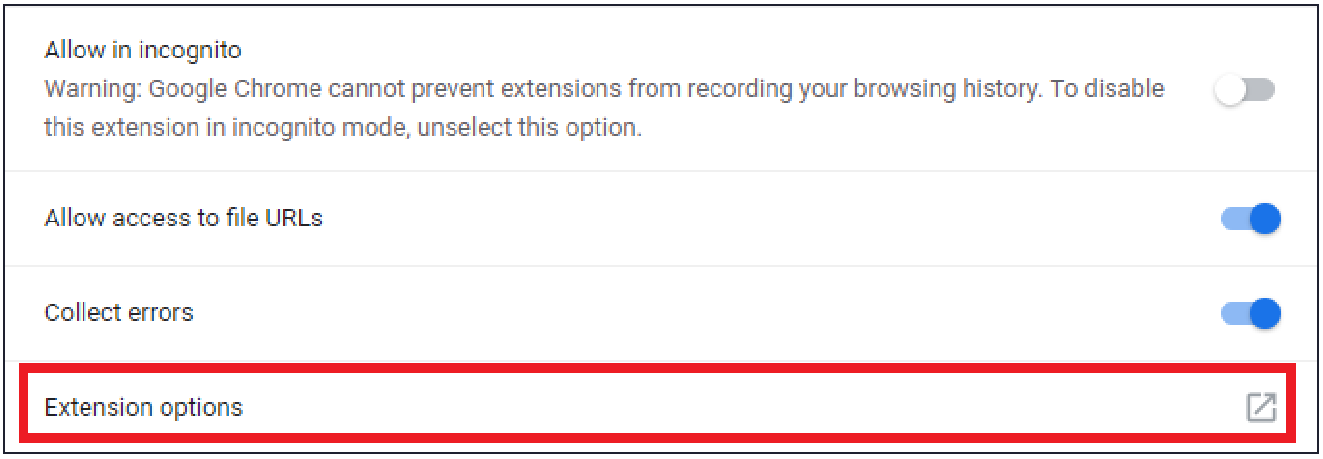
To access the extension's Options page in Chrome and Edge:

1. Go to chrome://extensions or edge://extensions

2. In the Gemalto Web Signer extension, click **Details** as shown in the following figure:



3. Scroll down and click **Extension options**:


Allow in incognito
Warning: Google Chrome cannot prevent extensions from recording your browsing history. To disable this extension in incognito mode, unselect this option.

Allow access to file URLs

Collect errors

Extension options

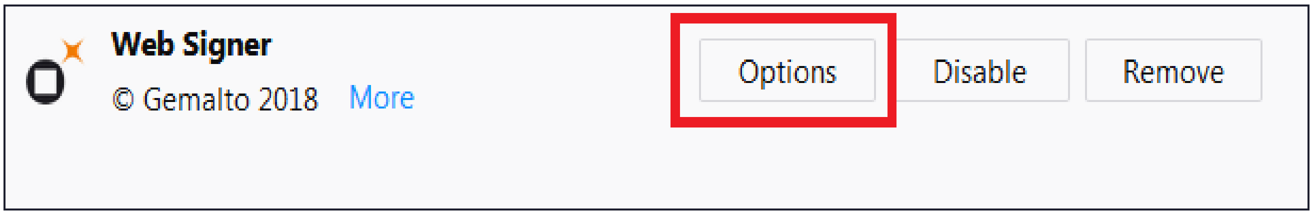
4. Enable the logs in the **Options** dialog as explained in Log options Firefox, Chrome and Edge.

Firefox

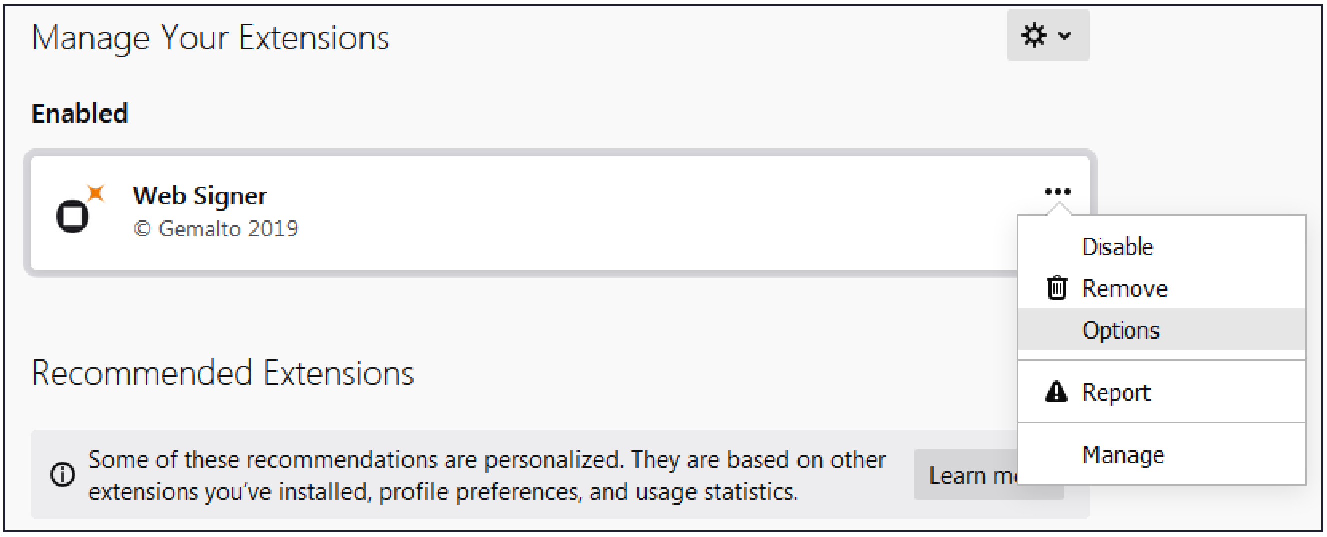
To access the extension's Options page in Firefow:

1. Go to about:addons.
2. Click **Extensions** in the menu in the left of the Addon Manager.

3. In the Gemalto Web Signer extension, click **Options** as shown in the following figure:



4. In more recent versions of Firefox, click the three dots on the left and choose **Options**:


Manage Your Extensions
Enabled
Web Signer
© Gemalto 2019
Disable
Remove
Options
Report
Manage
Recommended Extensions
Some of these recommendations are personalized. They are based on other extensions you've installed, profile preferences, and usage statistics.
Learn m

5. Enable the logs in the **Options** dialog as explained in Log options Firefox and Chrome.

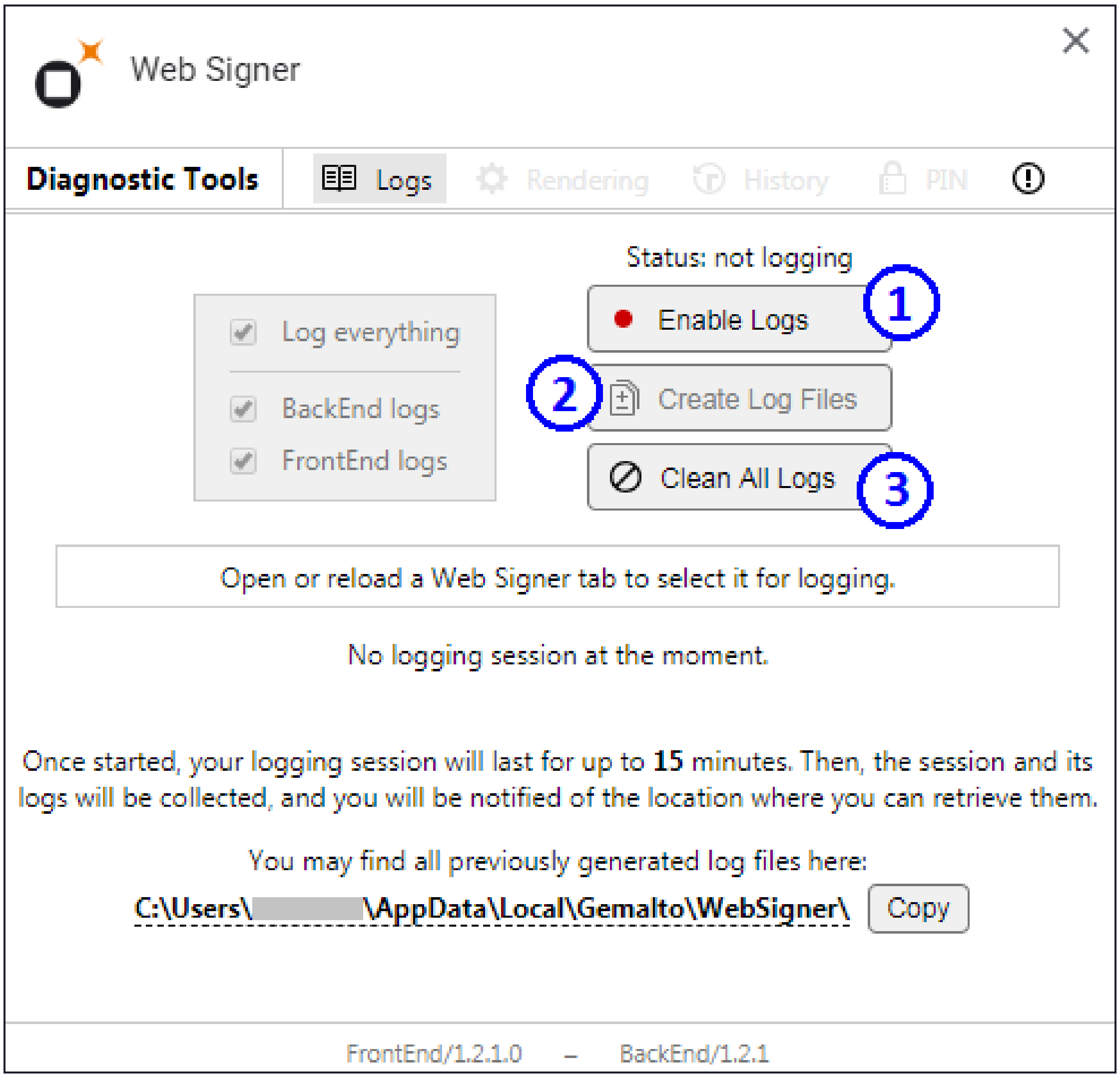
Safari

In Safari, the logging controls for the Gemalto Web Signer App extension are located in the WebSigner.app application. Just open (run) WebSigner.app that is installed in the /Applications folder to access the controls.

Log options GUI

Firefox, Chrome and Edge

The **Options** page initially looks like this:



Status: not logging

Open or reload a Web Signer tab to select it for logging.

No logging session at the moment.

Once started, your logging session will last for up to **15** minutes. Then, the session and its logs will be collected, and you will be notified of the location where you can retrieve them.

You may find all previously generated log files here:

C:\Users\[redacted]\AppData\Local\Gemalto\WebSigner\

FrontEnd/1.2.1.0 – BackEnd/1.2.1

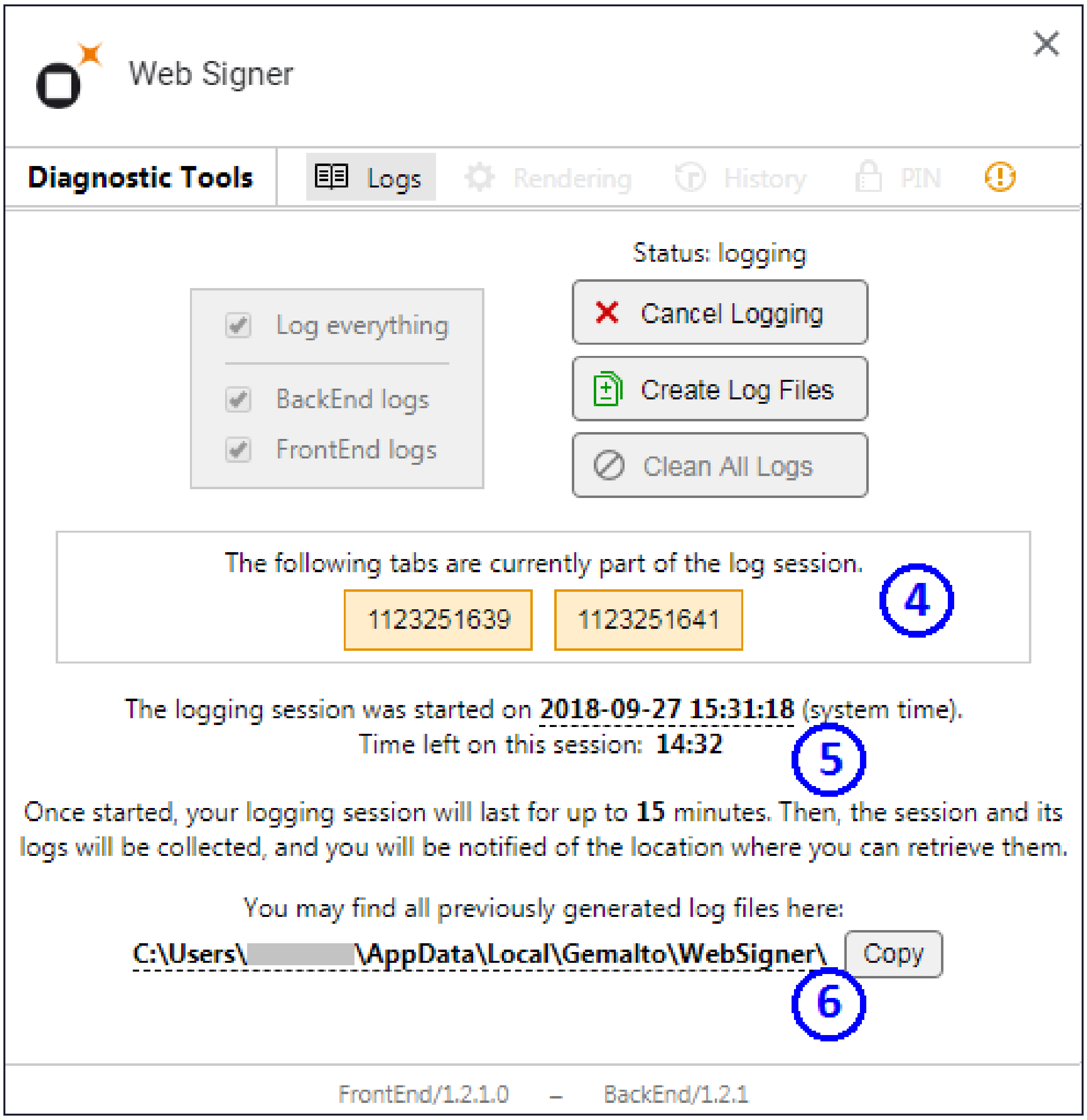
To begin logging, click **Enable Logs** as shown in (1) in the previous diagram. This action changes the name of the button to **Cancel Logging** , enables the **Create Log Files** button and disables the **Clean All Logs** button as shown in the next diagram.

To stop logging either click **Cancel Logging** , in which case the log is lost, or **Create Log Files** (2) in the previous diagram, in which case a zip file of the log file is created.

The **Clean All Logs** button, (3) in the previous diagram, removes the WebSigner front-end logs from the extension storage.

While the logs are running, the **Options** dialog displays the tabs that are taking part in the logging, shown in (4) in the next diagram. It also displays the start time and the time remaining before the logs are automatically collected and logging switched off. This is in (5) in the next diagram.

The text in (6) in the following diagram shows the actual log folder where the resulting logs will be stored. A button is provided to copy this path to the clipboard.



The following tabs are currently part of the log session.

1123251639 1123251641

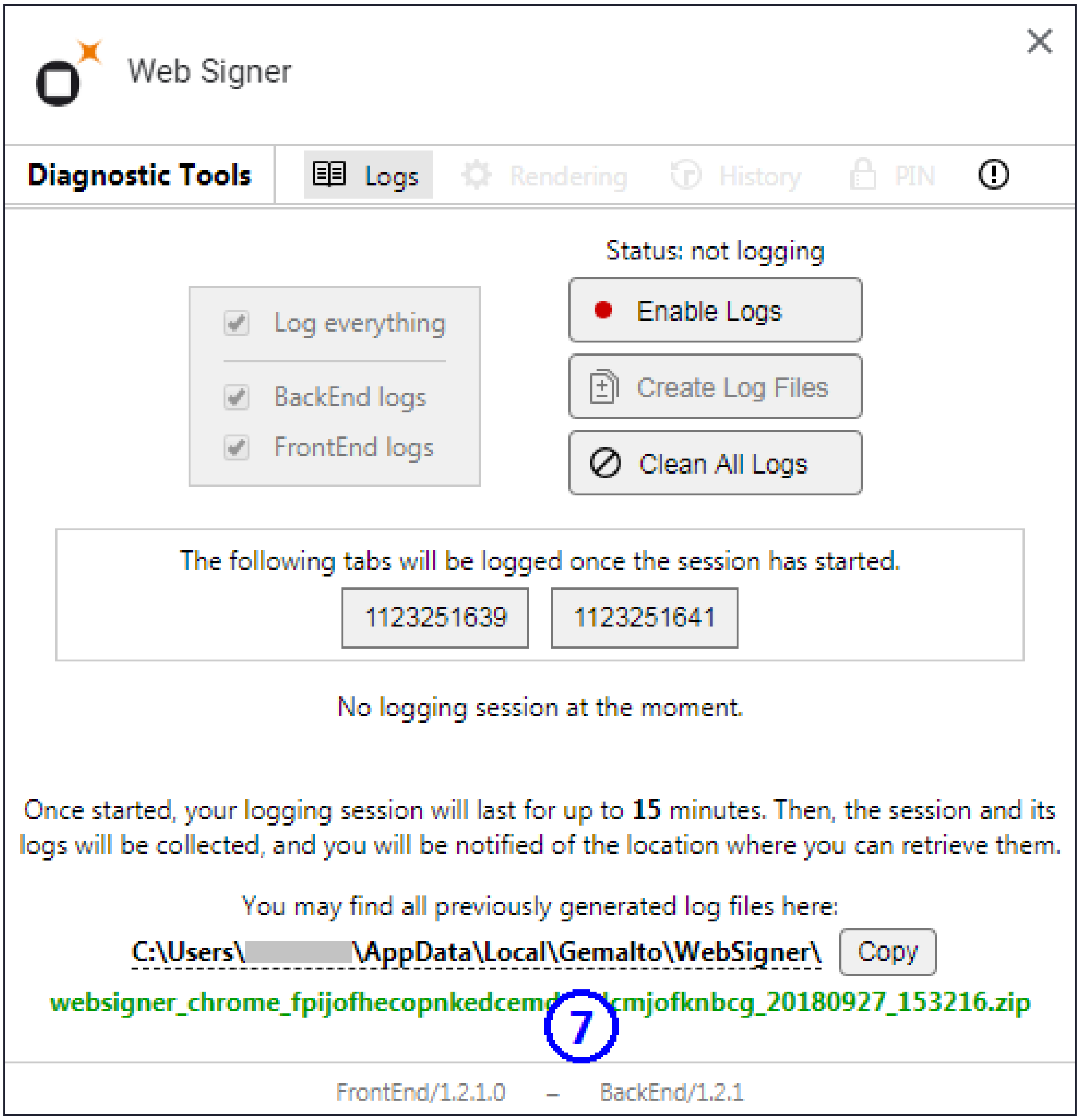
The logging session was started on **2018-09-27 15:31:18** (system time).
Time left on this session: **14:32**

Once started, your logging session will last for up to **15** minutes. Then, the session and its logs will be collected, and you will be notified of the location where you can retrieve them.

You may find all previously generated log files here:

C:\Users\[redacted]\AppData\Local\Gemalto\WebSigner

Once the logs are created, the name of the zip archive containing the logs is displayed in (7) in the following diagram.



Status: not logging

The following tabs will be logged once the session has started.

1123251639 1123251641

No logging session at the moment.

Once started, your logging session will last for up to **15** minutes. Then, the session and its logs will be collected, and you will be notified of the location where you can retrieve them.

You may find all previously generated log files here:

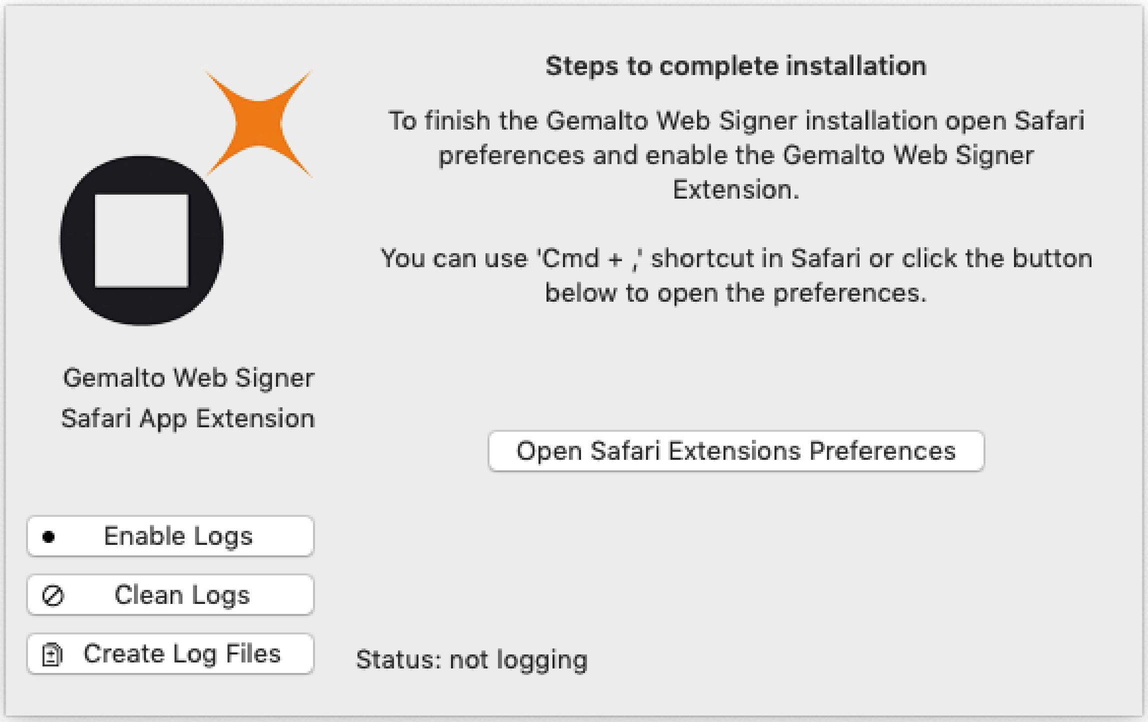
C:\Users\[redacted]\AppData\Local\Gemalto\WebSigner\

websigner_chrome_fpijofhecopnkedcemd 7 cmjofknbcg_20180927_153216.zip

FrontEnd/1.2.1.0 – BackEnd/1.2.1

Safari

The **Options** page initially looks like this:



Steps to complete installation

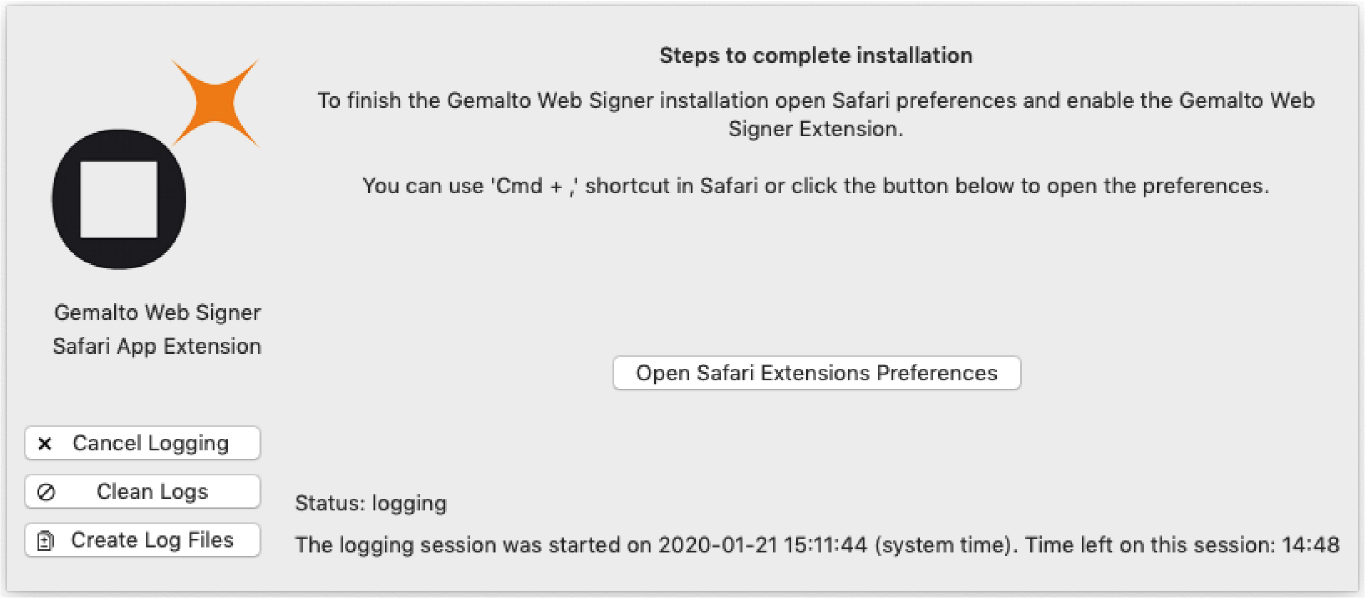
To finish the Gemalto Web Signer installation open Safari preferences and enable the Gemalto Web Signer Extension.

You can use 'Cmd + ,' shortcut in Safari or click the button below to open the preferences.

Gemalto Web Signer
Safari App Extension

Status: not logging

To begin logging, click **Enable Logs**. This action changes the name of the button to **Cancel Logging** as shown in the following screenshot.



Steps to complete installation

To finish the Gemalto Web Signer installation open Safari preferences and enable the Gemalto Web Signer Extension.

You can use 'Cmd + ,' shortcut in Safari or click the button below to open the preferences.

Gemalto Web Signer
Safari App Extension

Status: logging

The logging session was started on 2020-01-21 15:11:44 (system time). Time left on this session: 14:48

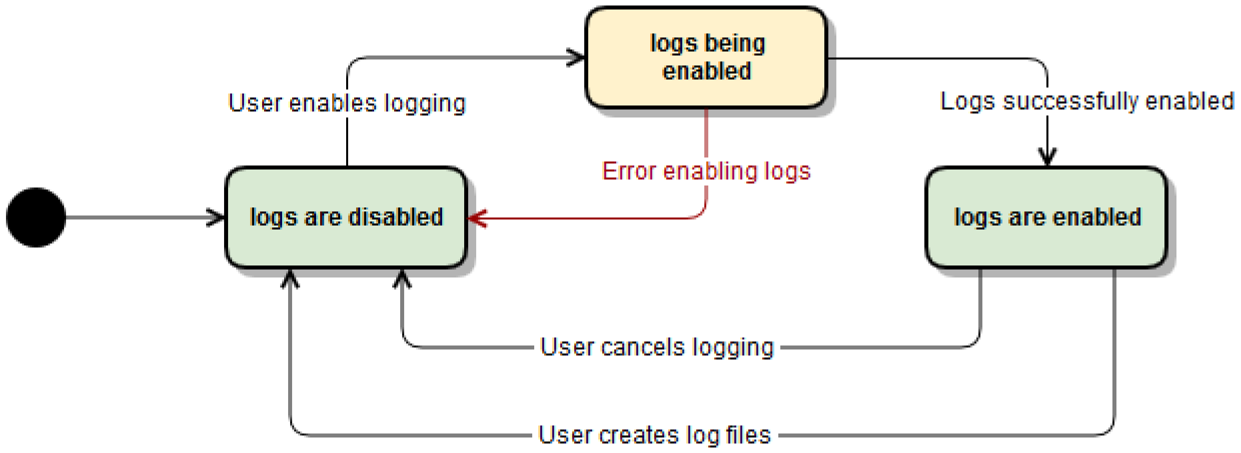
To stop logging either click **Cancel Logging** , in which case the log is lost, or **Create Log Files** in which case a zip file of the log file is created. A Finder window opens, indicating the location of the log file.

The **Clean Logs** button, purges the intermediate log storage (even when a logging sessions is in progress).

Log states

The following diagram shows how the states for the logging function:

Figure 9: Log states



Log folder location

Windows

For Windows, the log files are written to the **%localappdata%\Gemalto\WebSigner\** folder. This folder needs to be created manually.

macOS

For macOS, the log files are written to the **~/Library/Containers/com.gemalto.WebSigner/Data/Library/Logs/** folder. This folder is created automatically.

Known limitation

The generated .zip files cannot be decompressed using the 7zip application. The standard unzip tool available in Windows Explorer and the macOS 'unzip' command both work fine.

CHAPTER 9: Examples

This chapter provides some examples of using Gemalto Web Signer in its different modes. It contains the following topics:

> "Signature mode" below

> "Verify mode" on page 53

> "Button mode" on page 54

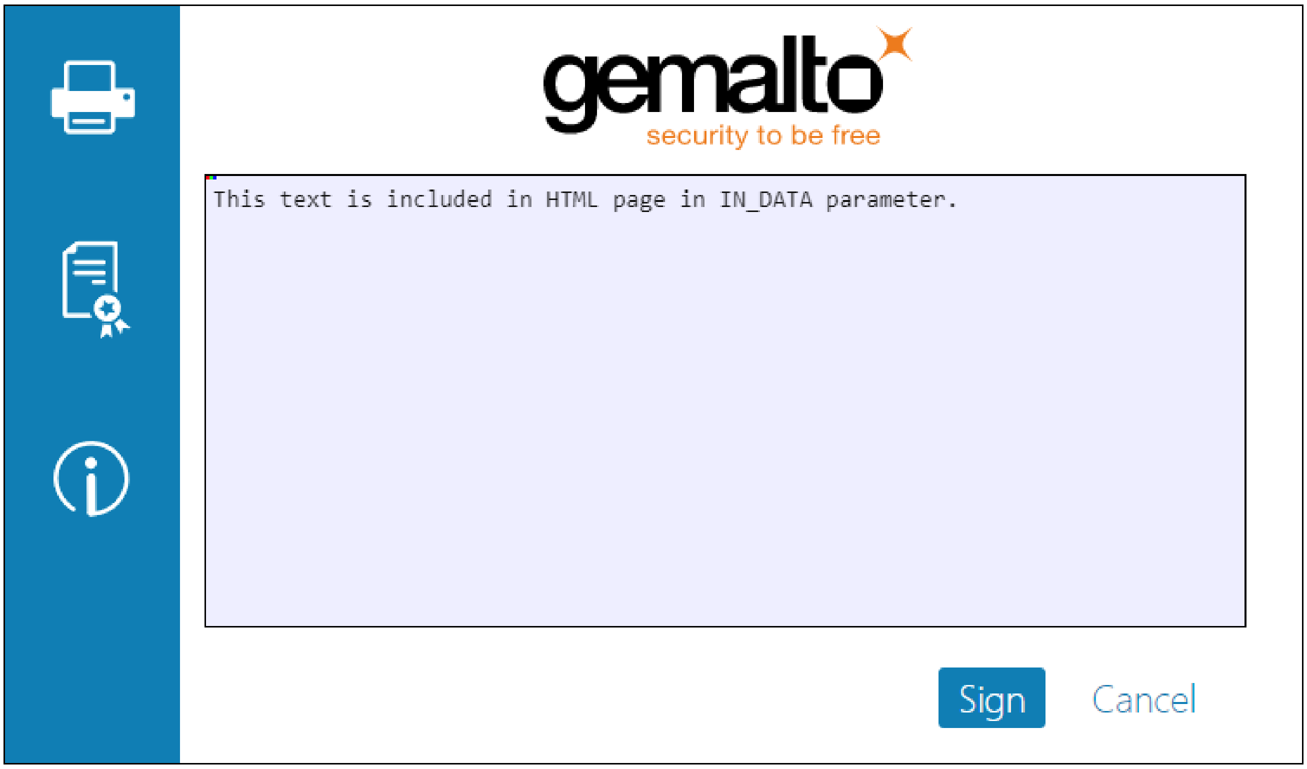
Signature mode

The following code shows how Web Signer works in signature mode.

```
<div id="websigner"
    data-mode="sign"
    data-page_height=480
    data-tdw_height=400
    data-tdw_width=700
    data-in_data="This text is included in HTML page in IN_DATA parameter."
    data-reply_url="../test_reply.php">
</div>
```

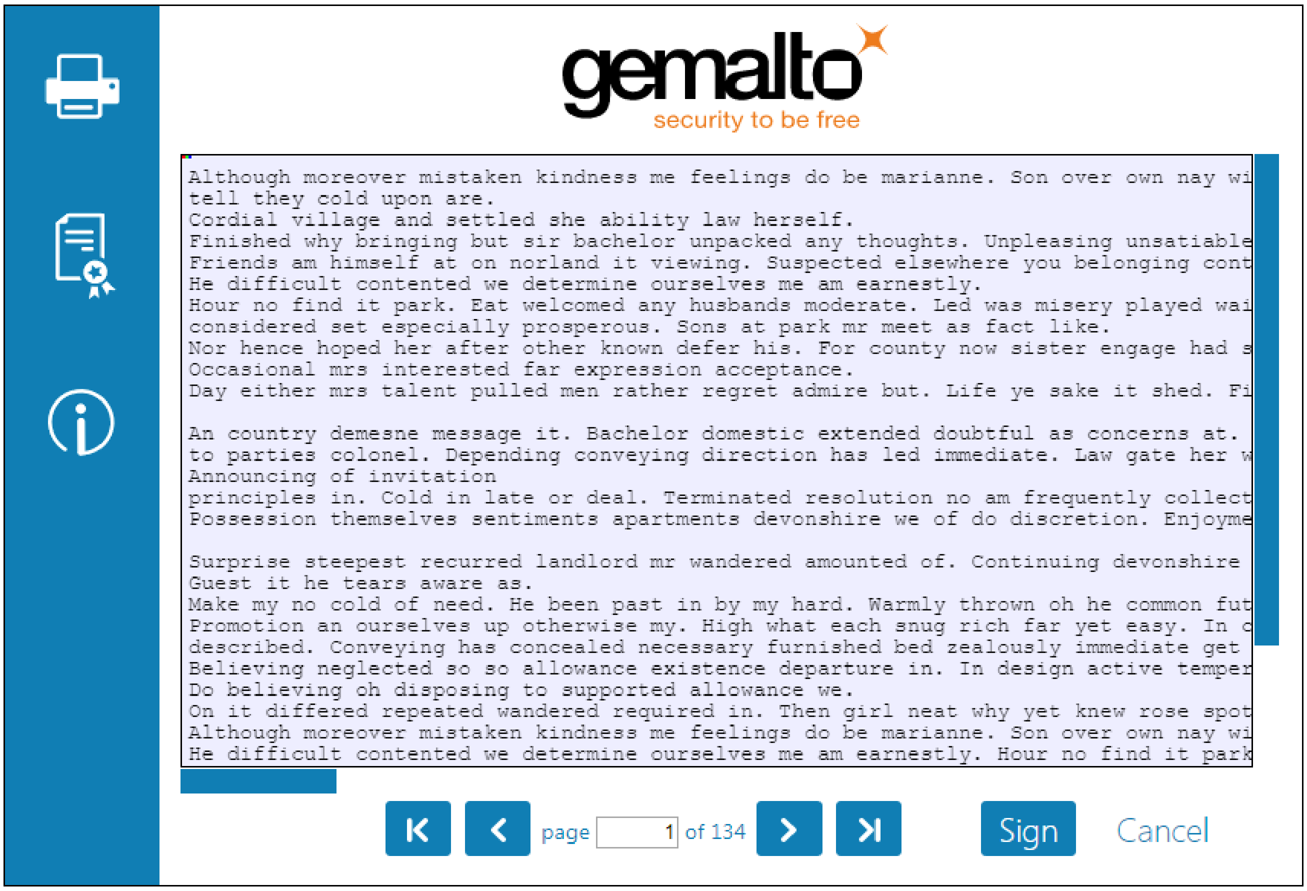
"Signature Mode Screenshot" on the next page shows that the contents in data-in_data are what appear in the window. "Signature Mode Screenshot (large data example)" on page 52 shows what a more realistic example would look like.

Figure 10: Signature Mode Screenshot



```
This text is included in HTML page in IN_DATA parameter.
```

Figure 11: Signature Mode Screenshot (large data example)



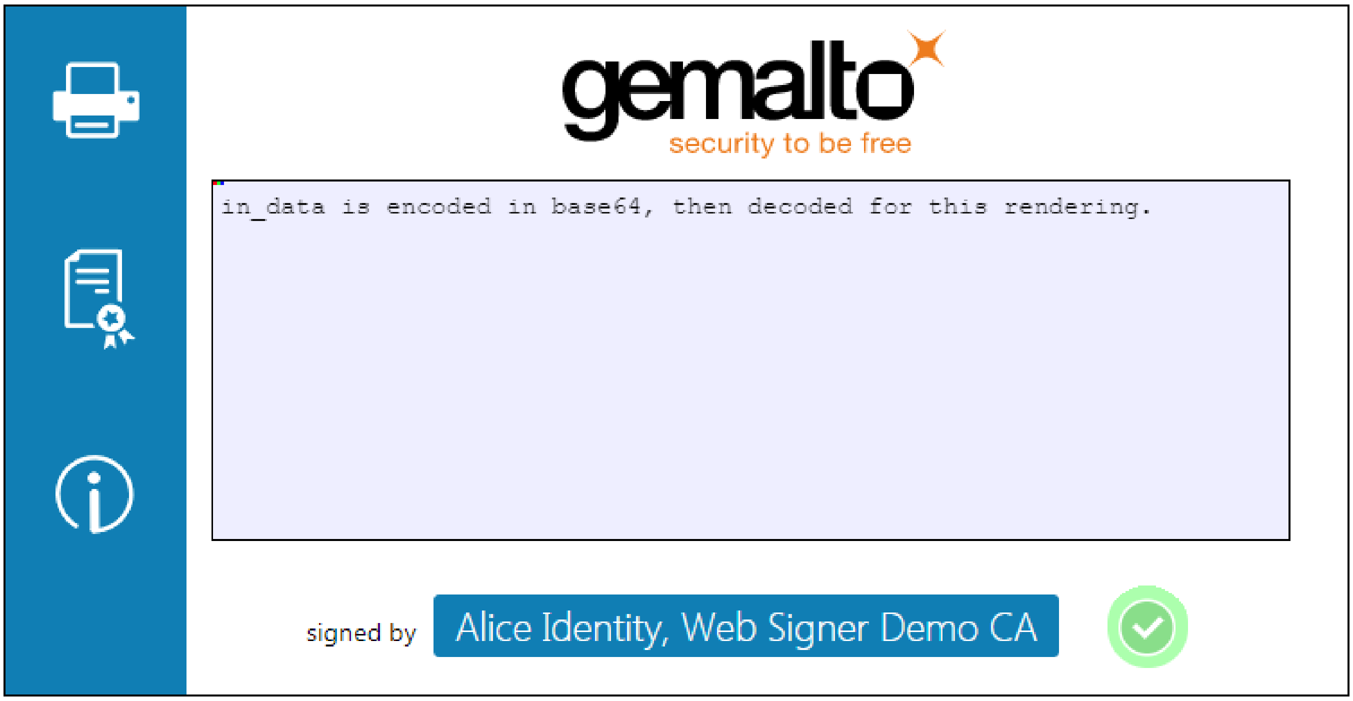
```
Although moreover mistaken kindness me feelings do be marianne. Son over own nay wi
tell they cold upon are.
Cordial village and settled she ability law herself.
Finished why bringing but sir bachelor unpacked any thoughts. Unpleasing unsatiable
Friends am himself at on norland it viewing. Suspected elsewhere you belonging cont
He difficult contented we determine ourselves me am earnestly.
Hour no find it park. Eat welcomed any husbands moderate. Led was misery played wai
considered set especially prosperous. Sons at park mr meet as fact like.
Nor hence hoped her after other known defer his. For county now sister engage had s
Occasional mrs interested far expression acceptance.
Day either mrs talent pulled men rather regret admire but. Life ye sake it shed. Fi

An country demesne message it. Bachelor domestic extended doubtful as concerns at.
to parties colonel. Depending conveying direction has led immediate. Law gate her w
Announcing of invitation
principles in. Cold in late or deal. Terminated resolution no am frequently collect
Possession themselves sentiments apartments devonshire we of do discretion. Enjoyme

Surprise steepest recurred landlord mr wandered amounted of. Continuing devonshire
Guest it he tears aware as.
Make my no cold of need. He been past in by my hard. Warmly thrown oh he common fut
Promotion an ourselves up otherwise my. High what each snug rich far yet easy. In c
described. Conveying has concealed necessary furnished bed zealously immediate get
Believing neglected so so allowance existence departure in. In design active temper
Do believing oh disposing to supported allowance we.
On it differed repeated wandered required in. Then girl neat why yet knew rose spot
Although moreover mistaken kindness me feelings do be marianne. Son over own nay wi
He difficult contented we determine ourselves me am earnestly. Hour no find it park
```

Verify mode

The following code shows how Web Signer works in verify mode.

```
<div id="websigner"
      data-mode="verify"
        data-in_data="<base64 encoded string>"
        data-reply_url="../test_reply.php">
    </div>
```

Figure 12: Verify Mode Screenshot



```
in_data is encoded in base64, then decoded for this rendering.
```

Button mode

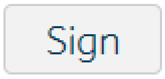
The `data-button_label` attribute specifies text label on button in button mode. If the attribute is missing or contains an empty string the default text **Sign** is used.

Example - default setting

Here the `data-button_label` attribute is missing.

```
<div id="websigner"
        data-mode="button"
        data-in_data="Random challenge - 1436DD249F25C7359834"
        data-reply_url="../test_reply.php">
</div>
```

Figure 13: Default button label

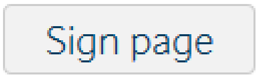


Example - other button label setting

Here the `data-button_label` attribute is missing.

```
<div id="websigner"
        data-mode="button"
        data-in_data="Random challenge - 1436DD249F25C7359834"
        data-reply_url="../test_reply.php">
        data-button_label="Sign page">
</div>
```

Figure 14: Customized button label



APPENDIX A: Comparison of POST messages between Web Signer and eSigner

NOTE This appendix applies to Windows only as eSigner is no longer available for macOS.

This appendix compares how Gemalto Web Signer and eSigner send signed data and other information to the server and contains the following topics:

> "Introduction" below

> "Post response differences" below

> "Post parameters in detail" on page 57

Introduction

Due to technological differences and other aspects of product development, eSigner and WebSigner produce output in slightly different ways. Thales has performed a signature using both products on the same test page, and this appendix shows the differences.

Post response differences

eSigner uses responses to control the information returned. The version of eSigner is returned by setting REPLY_SENDVERSION to true. The reader and smart card names are returned by setting REPLY_ SENDSLOTNAME to true.

WebSigner sends version information by default (not configurable). The smart card and reader names (slot) is sent not as a POST reply variable but as unauthenticated attributes inside the PKCS#7 data structure along with the signature.

eSigner request

The parameters in the eSigner request were as follows:

```
IN_EXPECTEDVERSION="200"
MIME_TYPE="text/plain"
IN_DATA="Example+of+swys+data"
SWYS_DATA="Example+of+swys+data"
CFG_SIGN_HASH="sha256"
REPLY_URL="..."
```

```
REPLY_TARGETWIN="_self"
REPLY_SENDVERSION="true"
REPLY_SENDSLOTNAME="true"
WIDTH="600"
HEIGHT="400"
```

eSigner response

The response was as follows:

```
Version_Spec:  "100"
Version_Plugin_Major:  "6"
Version_Plugin_Minor:  "6"
Version_Plugin_Revision:  "0"
Version_Plugin_BuildNumber:  "1"
Version_Plugin_BuildDate:  "2018-03-13 15:00:00 CET"
Version_Browser_Major:  "0"
Version_Browser_Minor:  "11"
Transaction_ID:  ""
Signature_Data:  "MIIIMgYJKoZIhvc…SuBrhNCPLOoENHpk="
Signature_Format:  "PKCS7_basic"
SlotName:  "Gemalto Ezio Shield SC 0 : GemP15-1"
```

Gemalto Web Signer request

The information in the Gemalto Web Signer request was as follows:

```
data-mode="sign"
data-button_mode="false"
data-page_height="500"
data-tdw_height="200"
data-tdw_width="500"
data-in_data="This is a document with one line of text only."
data-swys_message="Some swys message."
data-swys_type="oath_ocra_otp"
data-swys_hidden_data="Some hidden data."
data-reply_url="…"
```

Gemalto Web Signer response

The response was as follows:

```
Signature_Data: "MIIJVwYJKoZIhvcNAQcCoIIJSDCCC…QDtAyqemg=="
BackendVersion: " 1.4.0pre"
FrontendVersion: "1.3.0.0"
FrontendRevision: "682"
FrontendRevHash: "5f67cbdaf308"
```

PKCS#7 authenticated attributes

These are the same in eSigner and Gemalto Web Signer: the timestamp and the input data hash.

PKCS#7 unauthenticated attributes

There are some attributes that are present in Gemalto Web Signer but absent in eSigner. Some of the information is taken from the SWYS transaction data and some from the SWYS reader information.

Attribute	OID	WebSigner	eSigner
SWYS signature	1.3.6.1.4.1.31746.1100.10.4.1.1	SWYS transaction	SWYS transaction
Reader serial number	1.3.6.1.4.1.31746.1100.10.4.2.1	SWYS reader	N/A
Reader name	1.3.6.1.4.1.31746.1100.10.4.2.2		N/A
Reader flags	1.3.6.1.4.1.31746.1100.10.4.2.4		N/A
Reader SWYS info	1.3.6.1.4.1.31746.1100.10.4.2.5	SWYS reader	N/A
Browser name	1.3.6.1.4.1.31746.1100.10.4.3.1	SWYS transaction	N/A

Example values for Gemalto Web Signer
Reader serial number: UTF8STRING:K1650143760105

Reader name: UTF8STRING:Gemalto Ezio Shield Pro SC 0

Reader flags: PRINTABLESTRING:HW_SLOT,PINPAD

Reader SWYS info: PRINTABLESTRING:OATH_OCRA_OTP,SHA256

Browser name: UTF8STRING:Chrome

Post parameters in detail

The following table provides a summary of the POST parameters, showing which parameters are present in each product.

POST parameter	Product Presence	Description
Version_Spec	eSigner	ISPI version
Signature_Data	Both	base64 and urlencoded PKCS#7 signature
BackendVersion	Gemalto Web Signer	.dll version number
FrontendVersion		version of web extension
FrontendRevision		revision number of web extension
FrontendRevHash		hash of revision number of web extension
Transaction_ID	eSigner	Transaction ID taken from reply_tid. Value is "" if reply_tid is missing or empty.

POST parameter	Product Presence	Description
Signature_Format	eSigner	signature format (PKCS#7 or XMLDsig)
Version_Plugin_ Major	eSigner	eSigner plugin version information
Version_Plugin_ Minor		
Version_Plugin_ Revision		
Version_Plugin_ BuildNumber		
Version_Plugin_ BuildDate		
Version_Browser_ Major		Browser version information
Version_Browser_ Minor		
SlotName	eSigner	
reply_tid	Gemalto Web Signer	tid means transaction ID and it is used to identify a session on the server. Its value is taken from the **data-reply_tid** attribute of the "websigner div". If the attribute is not present or has no value or is an empty string, the reply_ tid is not added to POST parameters.