Application Note
Web Signing

Document version 1.1
31.10.2008

Population Register Centre (VRK)
Certification Authority Services
P.O. Box 70
FIN-00581 Helsinki
Finland
http://www.fineid.fi

Authors

Name	Initials	Organization
Timo Ukkonen	TUk	Fujitsu Services OY

Document history

Version	Date	Description	Editor
1.0	02.04.2008	First edition	TUk
1.1	31.10.2008	New line break processing (chapters 3.3.2 and 4.2), SetWeb compatible data encoding (3.3.2)	TUk

Contents

1 Introduction

1.1 Background

Population Register Centre (VRK) issues certificates stored on a FINEID smart card. As a part of the service offering VRK has compatible middleware (card reader software) available to the holders of a FINEID card.

1.2 Content overview

This document specifies the interface and the functionality of the HTTP signing module that is a part of the card reader software package. The HTTP signing module implements a browser and OS independent solution for making non-repudiation signatures from web applications.

Supported operating systems are:

Windows (x86-32 and x86-64 processor architectures)

Windows 2000 (only x86-32)

Windows XP

Windows 2003

Windows Vista

Linux (x86-32 and x86-64 processor architectures, GNOME user interface)

SUSE Linux Enterprise Desktop 10

Redhat Enterprise Linux 5

Ubuntu 7.10

MAC OS X (x86-32 processor architecture)

Version 10.4 (Tiger)

Version 10.5 (Leopard)

1.3 Intended audience

This document is intended for a technical audience. Understanding of certificates, smart cards and web application development is needed for best benefit of this documentation. The purpose of this document is to give system and application developers the necessary information to plan and implement solutions and services that utilize web signing within the FINEID context.

1.4 References

It is beneficial for the reader to be familiar with the FINEID specifications that may be downloaded from http://www.fineid.fi/vrk/fineid/home.nsf/en/documents

FINEID S1 - Electronic ID Application, v2.1 15.03.2004

FINEID S1 - Electronic ID Application, v1.12 04.11.2002

FINEID S2 - VRK (PRC) CA-model and certificate contents, v2.1 05.07.2005

FINEID S2 - VRK (PRC) CA-model and certificate contents, v2.0 24.03.2003

FINEID S4-1 - FINEID Implementation profile 1 for Finnish Electronic ID Card, v2.1A 22.10.2004

FINEID S4-1 - FINEID Implementation profile 1 for Finnish Electronic ID Card, v1.31 04.11.2002

FINEID S4-1 - FINEID Implementation profile 1 for Finnish Electronic ID Card, v1.1 02.12.1999

FINEID S4-2 - FINEID Implementation profile 2 for Organizational Usage, v2.1A 22.10.2004

FINEID S4-2 - FINEID Implementation profile 2 for Organizational Usage, v1.31A 05.12.2003

FINEID S4-2 - FINEID Implementation profile 2 for Organizational Usage, v0.9 01.03.2000

FINEID S5 - Directory Specification, v2.2 27.3.2007

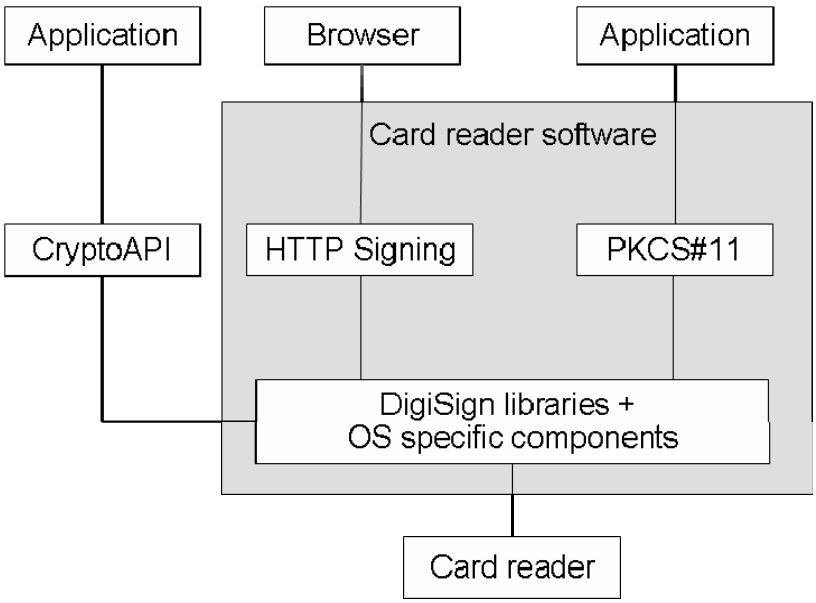
Guidelines for Developing Applications for FINEID card, v1.1

1.5 Terms and Definitions

Card reader software	A software package for smart card authentication, signing and encryption.
HTTP Signing module	Internal part of card reader software that implements signing for web applications.
HTTP	Hypertext Transfer Protocol. Unsecure communication protocol used the transfer information in World Wide Web.
SSL	Secure Sockets Layer, A Cryptographic protocol to provide secure communications on the internet.
HTTPS	HTTP over SSL. Secure HTTP communications.

2 Architectural overview

Overview of the DigiSign client architecture is following



Card reader software offers three different interfaces for accessing its services:

- OS specific cryptographic API (CryptoAPI)
- PKCS#11
- HTTP interface for signing.

Implementations of these interfaces are based on Fujitsu DigiSign libraries. These libraries implements common cryptographic operations and access to the smart card data and operations. There are also some operating system specific modules which offers access to the smart cards for the operating system and applications using operating systems API's.

3 The HTTP Signing Module

3.1 HTTP interface for signing

HTTP Signing module accepts HTTP 1.1 request either via unsecure (http) or secure (https) channel.

For HTTPS, only supported protocol is SSL 3.0 with RSA authentication and 3DES encryption (SSL Cipher suite 0x000A, which is widely supported by different browsers).

HTTP interface accepts two requests.

HTTP POST containing signing request (chapter 3.3)

Empty HTTP GET for checking if the module is running (chapter 3.4)

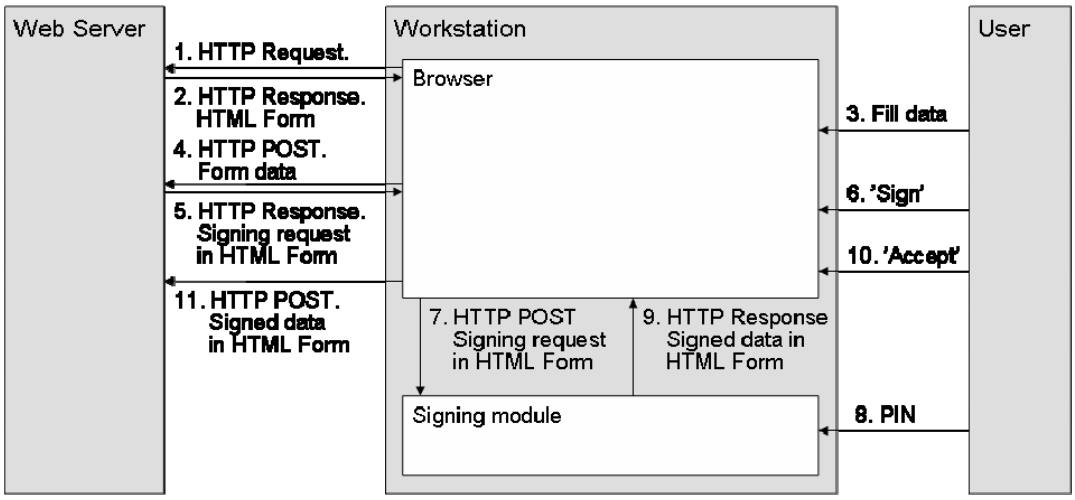
Requests are only accepted from the local workstation.

Data may be signed only with certificates whose key usage is 'Non-Repudiation'.

3.2 Overview of the HTTP Signing module

HTTP Signing module is a local HTTP server on the user's workstation that accepts only local request. The request is a HTTP POST request that contains a HTML form data. Signing request is stored on one hidden field of this form.

The following picture illustrates the co-operation of the components that are related to the signing process.



The control flow is following:

1. Browser sends a request to the web server by the user request.
2. Web server builds a html form where user can fill in the data to be signed. If the data source is not the user, operation is started from step 5.
3. User fills the data to the form and presses the submit button of the form.
4. Browser generates a request to the server.
5. Server processes the data and responds with HTTP response that has a signing request in a HTML form. For more details see next chapter.
6. User confirms the signing. If browser supports scripting this step may be omitted by implementing a script that posts the form to the signing module.
7. Browser posts the form data to the signing module.
8. Signing module parses the signing request.

   If certificates which are capable to non repudiation signatures are not found, an error page (chapter 3.3.3.2) will be returned. If only one certificate is found, it will be used automatically. Otherwise a certificate selection dialog will be displayed and the user must select the certificate.

   A signing dialog will be displayed with the data to be signed. For more information about the data requirements see chapter 3.3.2.

   User enters his PIN (or cancels the operation). Data will be signed.

9. Signing module composes a response for the request according to the signing request specification.
10. User submits the result back to the server. If the browser supports scripting, this phase may be omitted by implementing a script that posts the form to the server.
11. Browser generates request to the server with the signed data.

3.3 Signing Request

3.3.1 HTML page for the request

Signing request (for step 5 in previous picture) is a HTML form with following properties

- 'action' is "http://127.0.0.1: 53951/Sign" or "https://127.0.0.1: 53952/Sign"
- 'method' is "post"
- 'enctype' is "multipart/form-data"
- Has a field named 'SignRequest' that has the actual signing request as an XML structure.
- Page encoding must be either UTF-8 or ISO 8859-1
- Clients older that 1.3.2-32 supports only windows style (CR LF) line breaks. Later versions supports both windows and unix (LF) style line breaks.

Very simple signing request page could look like following:

```
<HTML>
    <body>
        <h2>Signing</h2>
        Insert the card to the reader and press OK!
        <form name="Form1" method="post"
            action="http://127.0.0.1:53951/Sign"
            id="Form1" enctype="multipart/form-data">
            <input type="hidden" name="SignRequest"
                value="[Signing request XML]"/>
            <input type="submit" name="Sign" value="OK"/>
        </form>
     </body>
</HTML>
```

The hidden input field named 'SignRequest' contains the signing request. It was not included on the example for readability reasons. It is described on the next chapter.

3.3.2 Signing request XML

The signing request is stored in a XML structure:

```
<SignRequest>
    <Data> </Data>
    <ResponsePage></ResponsePage>
    <ErrorPages>
        <Cancel></Cancel>
        <Other></Other>
    </ErrorPages>
</SignRequest>
```

XML elements are following:

- **SignRequest**: The root element for the request
- **Data**: Data to be signed as printable string. Up to 2 MB of data is accepted. For non-repudiation purposes data should contain only printable characters.

  If data contains non printable characters, data is processed as binary data and it will not be displayed in the signing dialog. In this case the signature may not be considered as a non repudiation signature.

  Line break processing has been changed in version 1.3.2-32. Earlier versions supports only Windows style line breaks (CR LF) and they remained same on the signed result. Current version supports both Unix (LF) and Windows (CR LF) style line breaks. All line breaks are converted to single LF to the signed result.

  Note: This element supports also old SetWeb compatible data encoding where ‘%’-character followed by hexadecimal digits are parsed as encoded characters. For example if sequence %25 is parsed as single ‘%’-character.
- **ResponsePage**: HTML template for the response after successful signing.
- **ErrorPages/Cancel**: HTML page for the when user cancelled the signing
- **ErrorPages/Other**: HTML page for other error cases. With current implementation this error page is used only when the request structure is invalid.

Example of the request:

```
<SignRequest>

    <Data><![CDATA[Data to Sign]]></Data>

    <ResponsePage><![CDATA[
        <html><body><h2>Signature</h2>
            <p>Data was signed successfully,
                press OK to accept it.</p>
            <form name="Result" method="post"
                action="http://some_url">
                <input type="hidden" name=" SignedData " value="%s"/>
                <input type="submit" name="Accept" value="OK"/>
            </form>
        </body></html>
    ]]></ResponsePage>
```

```
    <ErrorPages>
        <Cancel><![CDATA[
            <html><body><h2>Signature</h2>
                <p>Error : Signing was cancelled
                    or no card was present</p>
                <form name="SignCancel" method="get"
                    action="http://some_url">
                    <input type="submit" name="SignCancel"
                        value="Back"/>
            </form></body></html>
        ]]></Cancel>

        <Other><![CDATA[
            <html><body><h2>Signature Error</h2>
                <p>Unknown error on signing</p>
                <form name="SignError" method="get"
                    action="http://some_url">
                    <input type="submit" Name="SignError"
                        value="Cancel"/>
            </form></body></html>
        ]]></Other>
    </ErrorPages>
</SignRequest>
```

Request must be either ISO-8859-1 or UTF-8 encoded.

3.3.3 Responses

3.3.3.1 Success

The signing request contains a template for the response that is sent to the browser after successful signing. ‘%s’ is used in the template as place holder for the signed data. Template page must be designed so that it will send the signed data back to the server either automatically or by user actions.

The signed data is stored as base64 encoded.

Very simple response page could be like following:

```
<html>
    <body>
        <h2>Signature</h2>
        <p>Data was signed successfully, press OK to accept it.</p>
        <form name="Result" method="post" action="http://some_url">
            <input type="hidden" name="SignedData" value="%s"/>
            <input type="submit" name="Accept" value="OK"/>
        </form>
    </body>
</html>
```

In this example, the signed data will be put on the form field named ‘SignedData’. This example requires user to accept this data, but it is possible to avoid this by using scripting.

3.3.3.2 Errors

After some error, appropriate error page is selected from the request data. Three types of error pages are used

- If user cancels the operation or the card is not found, HTML definition for the error is read from **ErrorPages/Cancel**–element on the signing request. This page should define where the user is directed after the error.
- If some other error happens on the signing (should not happen if software is properly installed), HTML definition for the error page is read from the **ErrorPages/Other**–element on the signing request. This page should define where the user is directed after the error.
- If the whole signing request is invalid, a predefined error page will be displayed. Because the application context is unknown in this case, the error page only instructs user to use 'back' button to return to the previous page.

Very simple error response page could be like following:

```
<html>
    <body>
        <h2>Signature</h2>
        <p>Error : Signing was cancelled or no card was present</p>
        <form name="SignCancel" method="get"
            action="http://some_url">
            <input type="submit" name="SignCancel" value="Back"/>
        </form>
    </body>
</html>
```

3.4 Checking if the signing module is running

Existence of the signing module may be checked by sending empty HTTP GET request to the same URL as the signing request. Response for the request will be a 1x1 pixel invisible image. Thus <img>-elements 'onError' handler may be used the implement the error handler like in the following example:

```
<img src="http://127.0.0.1:53951/Sign"
    onError="alert('Signing module is not running')"/>
```

4 Security considerations

4.1 Protocols

To avoid warnings on the browser, signing requests should be made with the same protocol (http/https) that is used between server and browser.

4.2 Man in the middle attacks

Man in the middle attacks could be possible between web server and HTTP signing module in the user's workstation because the data is not ciphered on the browser.

For more details refer to chapter 3.2. Attack may happen in three different phases:

Between steps 3 and 4: Only the user can detect this situation and thus it is important that the signed data is printable and users are instructed to read the data in the signing dialog before signing it.

Between steps 5 and 7: Both user and server can detect this. Server side should compare the signed data to the data on the signing request to detect this. Note that due to line break processing (3.3.2) the signed data is not always exactly same as in the request.

Between steps 9 and 11: Only the server can detect this. In this case the signature is not valid at all.

5 Appendixes

5.1 Appendix A: A Test page for signing

Following HTML page is a test page for testing the signing functionality.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
   <HEAD><title>SignTest</title></HEAD>
   <body>
      <form name="Form1" method="post"
         action="https://127.0.0.1:53952/Sign" id="Form1"
         enctype="multipart/form-data">
         <textarea style="width:99%;height:550px" name="SignRequest">
<SignRequest>
   <Data><![CDATA[data to sign]]></Data>
   <ResponsePage><![CDATA[
      <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
      <html>
         <body>
            <h2>Fujitsu mPollux DigiSign Client</h2>
            <p>Signed data:<br/>%s</p>
         </body>
      </html>
   ]]></ResponsePage>
   <ErrorPages>
      <Cancel><![CDATA[
         <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
         <html>
            <body>
               <h2>Fujitsu mPollux DigiSign Client Error</h2>
               <p>Signing was cancelled or no card was present</p>
            </body>
         </html>
      ]]></Cancel>
      <Other><![CDATA[
         <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
         <html>
            <body>
               <h2>Fujitsu mPollux DigiSign Client Error</h2>
               <p>Unknown error on signing</p>
            </body>
         </html>
      ]]></Other>
   </ErrorPages>
```

```
</SignRequest>
       </textarea>
       <table width="100%" cellpadding="0" cellspacing="0">
         <tr>
           <td height="34" width="180"> 
             <input type="submit" name="Button3" value="Sign"/>
             <input type="reset" name="3" value="Reset"/>
           </td>
         </tr>
       </table>
     </form>
     <img src="https://127.0.0.1:53952/Sign"
       onError="alert('Signer is not running')"/>
  </body>
</HTML>
```