

Practical use of the OpenSign applets

Carsten Raskgaard <carsten@raskgaard.dk>

Monday, 22 Nov 2004

Introduction

During this presentation I will

- **Introduce the OpenOces project**
- **In particular introduce the OpenSign applets**
- **Show how the OpenSign applets work from a user's point of view**
- **Give a minimalistic example on how to integrate the applets into a website**
- **Describe how you build and sign the applets yourself**

Introducing OpenOces and OpenSign

Introducing OpenOces and OpenSign

Purpose of the OpenOces project

The purpose of the project is to ensure development of platform independent components for issuance and use of OCES certificates for citizens and employees according to the OCES Certificate Policies from the Danish National IT and Telecom Agency[..]

The project must ensure that the solutions are established and developed in an open environment with open source.

Solutions must in widest possible extends apply to both national and international de jure and de facto standards.

—A quote from www.openoces.org

A few details about the OpenOces project

- **Part of the OCES contract between TDC and the Ministry of Science, Technology and Innovation**
- **Homepage: www.openoces.org
[<http://www.openoces.org>]**
- **Organization**
 - **Coordination team**
 - **Core development team**
 - **The community**
- **Projects**
 - **OpenSign (signing and logon)**
 - **OpenCert (certificate issuance)**

The OpenSign project

OpenSign is a collection of Java applets providing client-side digital signing functionality using x.509 certificates. It currently consists of two applets, one for signing plain ASCII text and another providing login functionality.

- **Original author: Jens Bo Friis, IT-Practice**
- **Two applets:**
 - **OpenSign**
 - **OpenLogon**
- **License: LGPL**
- **Platforms: Linux, MacOS X, Microsoft Windows**
- **Browsers: Firefox, Galeon, Internet Explorer, Mozilla, Safari**
- **Vms: Microsoft, Blackdown, Sun**

The OpenSign project (Continued)

- **Keystores: through capi, pkcs#12**
- **Output format: xmldsig**
- **Sizes:**
 - **cab: 65kb**
 - **jar: 109kb**
 - **dll: 116kb**
- **Alternatives: ssl client authentication, capicom, cbt**

A user's perspective

A user's perspective

An example: A user selects a software license and signs it using the OpenSign applet

Try it yourself here [<http://www.openoces.org/demo>]

Using OpenSign on a website

Using OpenSign on a website

We will walk through all the steps needed to get the OpenSign applets up and running

Downloading OpenSign

- **Available from the homepage**
- **Both binary and source releases**
- **The applets in the binary releases have been signed by TDC**
- **Versioning style:**
 - **v1.0.x - stable**
 - **v1.1.x - unstable**
 - **v1.2.x - stable**

The dataflow

The applets operate in one of two modes

- **2-step mode**
 - **1st step: The applet is launched**
 - **2nd step: The applet exits and invokes a javascript method with the signed xmldsig document as argument**
- **3-step mode**
 - **1st step: The applet is launched**
 - **2nd step: The applet POSTs the generated xmldsig to an URL**
 - **3rd step: The user is forwarded to an URL**

2-step mode requires that liveconnect is supported by the browser. This has not until recently been the case for the Safari browser.

The dataflow (Continued)

When using 3-step mode the truststore of the jvm is used when establishing the ssl connection in the 2nd step. Ssl server certificates issued by TDC will therefore generate a warning message.

2-step mode: 1st step

```
<applet name="foo"  
  code="org.openoces.opensign.client.applet.Sign"  
  width="600" height="400" codebase="/demo"  
  archive="/demo/v1.2.2-1/OpenSign.jar" mayscript>  
  
<param name="locale" value="en,US">  
<param name="cabbage"  
  value="/demo/v1.2.2-1/OpenSign.cab">  
<param name="loglevel" value="debug">  
<param name="signtext"  
  value="RW50ZXIgdGV4dCB0byBiZSBzaWduZWQgaGVyZQ==">  
[...]  
</applet>
```

2-step mode: 2nd step

```
<script language="JavaScript">
function onSignOK(signature) {
    document.signedForm.message.value=signature;
    document.signedForm.result.value='ok';
    document.signedForm.submit();
}
function onSignCancel() {
    document.signedForm.result.value='cancel';
    document.signedForm.submit();
}
function onSignError(msg) {
    document.signedForm.result.value=msg;
    document.signedForm.submit();
}
</script>
```

2-step mode: 2nd step (Continued)

3-step mode: 1st step

```
<applet name="foo" [...]>
  [...]
  <param name="opensign.doappletrequest" value="true">
  <param name="opensign.verifieruri"
    value="/demo/servlet/Verifier?id=77">
  <param name="opensign.canceluri"
    value="/demo/servlet/Display?result=cancel">
  <param name="opensign.erroruri"
    value="/demo/servlet/Display?result=error">
  <param name="opensign.alerturi"
    value="/demo/servlet/Display?result=alert">
  <param name="opensign.verifiedokuri"
    value="/demo/servlet/Display?result=ok&id=77">
  <param name="opensign.verifiederroruri"
    value="/demo/servlet/Display?result=error&id=77">
  <param name="opensign.message.name" value="message">
  <param name="opensign.result.name" value="result">
```

3-step mode: 1st step (Continued)

```
[..]  
</applet>
```

3-step mode: 2nd step

```
public class Verifier extends HttpServlet {
    public void doPost(..)
    try {
        String xmldoc, xmldoc_b64;
        xmldoc_b64 = request.getParameter("message");

        xmldoc = new String(Base64.decode(
            xmldoc_b64.getBytes("UTF-8")), "UTF-8");
        Document doc = XML.getDocument(xmldoc);

        if (XML.verifySignature(doc)) {
            // do something
        } else throw new Exception();
        ..
    }
}
```

3-step mode: 2nd step (Continued)

3-step mode: 3rd step

The user is simply directed to the specified URL depending on the result of the verification in the previous step.

Structure of the generated xmldsig document

```
<?xml version="1.0" encoding="UTF-8" ?>
<openoces:signature [..]>
  <ds:Signature [..]>
    <ds:SignedInfo [..]>[..]</ds:SignedInfo>
    <ds:SignatureValue>[..]</ds:SignatureValue>
    <ds:KeyInfo>[..]</ds:KeyInfo>
    <ds:Object [..]>
      <ds:SignatureProperties>
        <ds:SignatureProperty><Name>signtext</Name>
          <Value VisibleToSigner="yes">[..]</Value>
        </ds:SignatureProperty>
      </ds:SignatureProperties>
    </ds:Object>
  </ds:Signature>
</openoces:signature>
```

Structure of the generated xmldsig document

Xmldsig signature validation

- **Signature validation is not part of the OpenSign project**
- **Easy to get wrong**
- **You must at least check that**
 - **the expected text was signed**
 - **the signature on the document is valid**
 - **the certificate is still valid**
 - **you trust the signer of the signing certificate**
 - **the certificate has not expired**
 - **the certificate has not been revoked**
 - **the certificate may be used for creating digital signatures**

Adding invisible elements to the signature

```
<applet [..]>  
[..]  
<param name="opensign.formdata.count" value="1">  
<param name="opensign.formdata.1.name" value="foo">  
<param name="opensign.formdata.1.value" value="bar">  
[..]  
</applet>
```

Adding invisible elements to the signature

```
[..]  
<ds:Object [..]>  
  <ds:SignatureProperties>  
    <ds:SignatureProperty><Name>foo</Name>  
      <Value VisibleToSigner="no">[..]</Value>  
    </ds:SignatureProperty>  
  </ds:SignatureProperties>  
</ds:Object>  
[..]
```

Important security issues

To protect against man-in-the-middle attacks:

```
<param name="logonto" value="some service">
```

Check that this value matches serverside

To protect against replay attacks:

```
<param name="opensign.formdata.1.challenge"  
value="Some large random value">
```

- Check that this value matches serverside
- use `java.security.SecureRandom` not `Math.random()`

Signing the applets yourself

- To get rid of the "signed by TDC" at startup
- Our experience: Thawte certificates work flawlessly

Signing the jar version:

```
$ ant prepareforsigningjar
$ jarsigner -keystore cert.p12 -storetype pkcs12 \
  -signedjar OpenSign.jar \
  dist/applet/unsigned/OpenSign.jar alias
```

Signing the cab version:

Signing the applets yourself (Continued)

```
$ ant prepareforsigningcab
$ mkdir tmp
$ cd tmp
$ unzip ../dist/applet/unsigned/OpenSign.zip
$ CabArc -r -p N OpenSign.cab *license *version \
  *class
$ signcode.exe -j ./javasign.dll -jp low -spc \
  ../cert.spc -v cert.pvk -n "OpenSign" \
  OpenSign.cab
$ signcode -x -t \
  http://timestamp.verisign.com/scripts/timestamp.dll
  -tr 5 OpenSign.cab
```

Epilogue

Epilogue

More information

- **The online documentation**
- **The demo setup at www.openoces.org - including the source code**
- **The mailinglists**
 - `<announce-general@openoces.org>`
 - `<development-announce@openoces.org>`
 - `<user@openoces.org>`
 - `<developer@openoces.org>`
 - **Also take a look in the archives**

**This presentation is available here
[<http://www.openoces.org/presentations>]**

The end

Software is like sex: it's better when it's free.

—Linus Torvalds

Thanks for listening!

Questions?