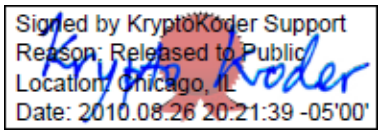
Signing a PDF document with PHP, Ruby or Python

MyPDFSigner provides extensions for PHP, Ruby and Python to sign PDF documents using tokens stored in PKCS#11 and PKCS#12 certificate stores. The extensions support timestamping (RFC 3161) and visible signatures, including the incorporation of "watermark" images. See example below.

Quick Start Guide



Install desktop application (Fedora|Ubuntu):
-- rpm -ihv mypdfsigner-1.4.2-1.x86_64.rpm
-- dpkg -i mypdfsigner_1.4.2-1_amd64.deb
Test command line:
-- mypdfsigner -i /usr/local/mypdfsigner/tests/example.pdf -o /tmp/example-signed.pdf -z /usr/local/mypdfsigner/tests/mypdfsigner.conf -v -q
Check /tmp/example-signed.pdf

Install extension (Fedora|Ubuntu):
-- rpm -ihv mypdfsigner-[php|ruby|python]-0.9.6-1.x86_64.rpm
-- dpkg -i mypdfsigner-[php|ruby|python]_0.9.6-1_amd64.deb
Test a script from the command line:
-- [php|ruby|python] /usr/local/mypdfsigner/tests/test.[php|rb|py]
Check /tmp/example-signed-[php|ruby|python].pdf

Note: MyPDFSigner for Linux provides two command line tools that are very similar, mypdfsigner-cli and mypdfsigner. The first uses iText Java code to do the signing (the same code used by the graphical application) while the second uses KryptoKoder's C code that is also used by the extensions.

Configuration

Before using any of the extensions it is necessary to start with the graphical application to create a configuration file for the key store and alias one wants to use. The application creates a .mypdfsigner file in your home directory. This file can be copied to /usr/local/mypdfsigner and renamed mypdfsigner.conf (this step can be skipped but then the configuration file needs to be specified when calling the "sign" function).

The extensions support PKCS#12 and PKCS#11 key stores. Use of PKCS#11 key stores is achieved through the OpenSSL engine cryptographic module. To install:
-- yum install engine_pkcs11 (Fedora)
-- apt-get install engine_pkcs11 (Ubuntu)

PKCS#12: For a PKCS#12 key store the configuration file created by the graphical application is ready to be used by the extensions.

PKCS#11: For a PKCS#11 key store some extra preparation and editing is required. By default, and unlike the PKCS#12 case, the PIN that protects the private key is not saved to the configuration file by the graphical application. However this can be changed by adding the entry "savepin=on" to the configuration file previously created (with the graphical application) and by signing a PDF document. This will prompt for the PIN

2014-03-25
Released PHP/Ruby/Python modules 0.9.6; adds support for updated PDFs.

2013-09-21
Released PHP/Ruby/Python modules 0.9.5; adds support for certified signatures.

2013-01-24
Released MyPDFSigner 1.4.2, a maintenance release.

2012-08-04
Released MyPDFSigner 1.4.0, a maintenance release; adds support for password protected PDFs.

2011-06-14
Released MyPDFSigner 1.3.0 with enhanced visible signatures configuration.

2011-04-27
Released MyPDFSigner 1.2.1 with support for OpenSC and PKCS#11 token based batch signing. **Permite assinatura em massa com o Cartão de Cidadão.**

2011-01-13
Released MyDWFSigner 0.8.0, a tool to sign DWF documents.

2010-09-24
Released MyPDFSigner 1.1.6 with support for configurable visible signatures.

2010-06-16
Released MyPDFSigner 1.1.0 with support for Time Stamping.

2010-04-14
Released MyPDFSigner 1.0.5 for Mac OS X only with support for the Apple Keychain Store.

which will then be encrypted and saved to the configuration file. Besides that a few other entries will be added to the file:

- signerpem: this is the path to a file with the certificate of the signer in PEM format
- capem: this is the path to a file with the chain of certificates in PEM format
- engine: the path to the engine_pkcs11.so file
- certid: the id of the alias to use; needs to be set manually

The signerpem and capem entries point to files in the user home directory. They can be moved elsewhere and the paths updated. Note that for the capem file to be correct the Certificates Store directory needs to be correct. This is the directory where the certificates that belong to the chain of trust of the siging certificate are placed (these certificates are usually not inside the PKCS#11 security device but instead are part of the software that comes with the device).

The path of the engine module defaults to /usr/lib/engines/engine_pkcs11.so if the module exists there. Otherwise it is left unfilled and needs to be filled manually. The certid needs to be set manually. To that end run the pkcs11-tool command (adapt to your PKCS#11 module and slot):
-- pkcs11-tool --module /usr/lib/opensc-pkcs11.so --slot-index 1 -O

The output is similar to the one below. The ID is the value that needs to be set in the certid entry. That would be certid=0eafe0cce98e9bb3f73cb6cd6b1399225be62492 in the example below.

```
Using slot with index 1 (0x1)
Certificate Object, type = X.509 cert
  label:      /CN=KryptoKoder Support/emailAddress=support@kryptokoder.com
  ID:         0eafe0cce98e9bb3f73cb6cd6b1399225be62492
Public Key Object; RSA 2048 bits
  label:      /CN=CAcert WoT User/emailAddress=support@kryptokoder.com
  ID:         0eafe0cce98e9bb3f73cb6cd6b1399225be62492
  Usage:      encrypt, verify
```

Once the configuration is ready you can test it using the command line. Make sure that signing with the command line works before trying out the extensions.

Timestamping: MyPDFSigner supports the HTTP(S) POST TSA protocol. Configuration is straightforward and is done with the entries 'tsaurl', 'tsauser' and 'tsapasswd' in the configuration file. If using HTTPS, a extra entry, 'tsacert' may be needed.

To establish a connection to a HTTPS Timestamping server, the CA that issued the certificate of the server needs to be trusted. If the CA certificate is already present in the ca-bundle.crt file (usually in the /etc/ssl/certs/ directory) then the trust is already established and no further steps are needed.

If trust has not been established then one needs to import the certificate of the CA that issued the timestamping server certificate. The certificate can be appended to the ca-bundle.crt file, or can be placed into its own file, whose path then needs to be used as the 'tsacert' entry in the configuartion file. In either case, the certificate needs to be converted to the PEM format:

```
$ openssl x509 -in CAcert -text -out CAcert.pem
```

Note: the above assumes the certificate in the CAcert file was already in PEM. If it is in DER format, then you need to add '-inform DER' to the above command.

2010-03-19
Released MyPDFSigner 1.0.0 with support for PKCS#12 files and a command line interface.

2009-12-05
Released MyPDFSigner 0.9.5 for Windows only with support for the Windows Certificate Store.

2009-11-10
Released MyPDFSigner 0.9.0 with support for all PKCS11 cards.

2009-10-19
Released MyPDFSigner 0.8.0 with support for the Portuguese Citizen Card only. **Funciona com o Cartão de Cidadão.**

You can test that the certificate is correct and that you can establish a connection to the server by using the command curl:

```
$ curl -v --cacert CAcert.pem https://tsa.example.com
```

Visible Signatures: MyPDFSigner supports visible signatures and allows for some customization (image, size, position and page). A signature is made visible by setting TRUE the "visible" argument of the "sign" function (or by passing -v when using the command line).

The visible signature is placed by default on the first page of the document. To place it in a different page add the entry sigpage=page to the configuration file. The "page" value is a positive or negative integer; if negative it means the pages are counted from the end. For instance, to place the signature on the last page the entry can just be sigpage=-1.

Before explaining how signature customization is done one needs to know about PDF size units, also known as points. Point units are based on a "72 units per inch" scale. Hence letter size (8.5 by 11 inches) corresponds to 612 x 792 points, and A4 (210 by 297 mm) corresponds to 595 x 842 points. A visible signature position is specified by an array of four values corresponding to the "lx" (left x), "by" (bottom y), "rx" (right x) and "ty" (top y) coordinates of the sides of the signature rectangle. MyPDFSigner can accept both positive and negative coordinates, with negative coordinates being mesured from the right and top edges of the page (positive units are measured from the lower left corner of the page). By default MyPDFSigner uses the rectangle [-170 -80 -40 -40]. A different signature rectangle can be specified in the configuration file by adding the entry sigrect=[lx by rx ty] with the new values. The text part of the visible signature consists of four lines: the cn (common name, obtained from the certificate), reason, location and date. The font size is adapted to the specified rectangle height. However the width of the rectangle needs to be manually tuned so that the text fits.

A visible signature can also incorporate an image if the "sigimage" entry is present in the configuration file. The image will be scaled to fit inside the signature rectangle. The suggested approach to select the right sized signature image is to start by choosing the right visible signature rectangle size (as explained above). Once that is known create a signature image with the same proportions (but high enough resolution so that it looks fine when printed). An example: if the rectangle is 130 (units) wide by 40 tall (like the default one) the size when printed will be 130/72 inches by 40/72 inches. Sign your name in a piece of paper inside a rectangle of such dimensions and scan it at, say, 300 dpi. This will create an image that is 130*300/72 (pixels) wide by 40*300/72 tall (or 542 by 167 pixels). An image with such resolution will scale nicely and the resulting graphics will have the right size. Images of different proportions can be used but since they will be scaled (and centered) to fit inside the rectangle there will be space around two of the sides of the image.

MyPDFSigner has some limitations regarding the type of images that can use. The image needs to be of RGB-Alpha PNG type. That is not a serious limitation since it is the default format used by Gimp when saving a color PNG file that includes a transparent layer. In case of doubt look at the image properties using Gimp.

An example of a signed and timestamped PDF document with a visible signature is available **here**. Note that it was signed with a self signed certificate so the warning one sees when opening in Adobe Reader is expected.

MyPDFSigner cannot yet handle PDF documents with compressed object streams.

These documents, which are not very common, are generally generated by Acrobat or similar.

MyPDFSigner API

MyPDFSigner does just one thing: it signs PDF documents. As such it provides just one function. Examples of its usage are shown next.

PHP Example

```
<?php

$originalPath = "/tmp/test.pdf";
$signedPath = "/tmp/test-signed.pdf";
$location = "Chicago";
$reason = "Testing";
$contactInfo = "support@kryptokoder.com";
$certify = TRUE;
$visible = TRUE;
$author = "KryptoKoder";
$tile = "Signing with MyPDFSigner";
$subject = "PHP Extension";
$keyowrds = "KryptoKoder, PKCS#12, PDF";
$confFile = ""; // defaults to /usr/local/mypdfsigner/mypdfsigner.conf if empty
$timestamp = TRUE;

echo mypdfsigner_sign($originalPath, $signedPath, $location, $reason, $contactInfo,
    $certify, $visible, $author, $title, $subject, $keywords, $confFile, $timestamp);

?>
```

Ruby Example

```
require 'mypdfsigner'
include MyPDFSigner

inputPath = "/tmp/input.pdf"
outputPath = "/tmp/output.pdf"
location = "Chicago"
reason = "Demo"
contactInfo = "+1 555-555-5555"
certify = true
visible = true
title = "Signing with MyPDFSigner"
author = "KryptoKoder"
subject = "Ruby Extension"
keywords = "PKCS#12, MyPDFSigner, PDF"
confFile = ""   # defaults to /usr/local/mypdfsigner/mypdfsigner.conf if empty
timestamp = true

puts mypdfsigner_sign(inputPath, outputPath, location, reason, contactInfo,
    certify, visible, title, author, subject, keywords, confFile, timestamp)
```

Python Example

```
import mypdfsigner

inputPath = "/tmp/input.pdf"
outputPath = "/tmp/output.pdf"
location = "Chicago, Illinois"
```

```
reason = "Demo"
contactInfo = "+1 555-555-5555"
certify = True
visible = True
title = "Signing with MyPDFSigner"
author = "KryptoKoder"
subject = "Python Extension"
keywords = "PKCS#12, PDF, MyPDFSigner"
confFile = ""   # defaults to /usr/local/mypdfsigner/mypdfsigner.conf if empty
timestamp = True

print mypdfsigner.sign(inputPath, outputPath, location, reason, contactInfo,
    certify, visible, title, author, subject, keywords, confFile, timestamp)
```

Remark: Note that although the path to the configuration file can be passed as an argument of the sign function, that approach is not recommended if using PKCS#11 key stores. Instead it is recommended that the configuration file is saved to the default location (/usr/local/mypdfsigner/mypdfsigner.conf) and an empty argument is passed to the sign function. This has the benefit that the registration of the PKCS#11 engine happens at startup time (i.e., when the web server starts) and the cleanup happens at shutdown time (when the web server shuts down). This issue is not relevant if using the command line or if using a PKCS#12 key store.

© 2009-2014 KryptoKoder