PDF encryption.

$Revision: 562881 $

by J.Pietschmann, Jeremias Märki

Table of contents

1. Overview

FOP supports encryption of PDF output, thanks to Patrick C. Lankswert. This feature is commonly used to prevent unauthorized viewing, printing, editing, copying text from the document and doing annotations. It is also possible to ask the user for a password in order to view the contents. Note that there already exist third party applications which can decrypt an encrypted PDF without effort and allow the aforementioned operations, therefore the degree of protection is limited.

For further information about features and restrictions regarding PDF encryption, look at the documentation coming with Adobe Acrobat or the technical documentation on the Adobe web site.

2. Usage (command line)

Encryption is enabled by supplying any of the encryption related options.

An owner password is set with the `-o` option. This password is actually used as encryption key. Many tools for PDF processing ask for this password to disregard any restriction imposed on the PDF document.

If no owner password has been supplied but FOP was asked to apply some restrictions, a random password is used. In this case it is obviously impossiible to disregard restrictions in PDF processing tools.

A user password, supplied with the `-u` option, will cause the PDF display software to ask the reader for this password in order to view the contents of the document. If no user password was supplied, viewing the content is not restricted.

Further restrictions can be imposed by using the `-noprint`, `-nocopy`, `-noedit` and `-noannotations` options, which disable printing, copying text, editing in Adobe Acrobat and making annotations, respectively.

3. Usage (embedded)

When FOP is embedded in another Java application you need to set an options map on the renderer. These are the supported options:

Option	Description	Values	Default
ownerPassword	The owner password	String	

userPassword	The user password	String	
allowPrint	Allows/disallows printing of the PDF	"TRUE" or "FALSE"	"TRUE"
allowCopyContent	Allows/disallows copy/paste of content	"TRUE" or "FALSE"	"TRUE"
allowEditContent	Allows/disallows editing of content	"TRUE" or "FALSE"	"TRUE"
allowEditAnnotations	Allows/disallows editing of annotations	"TRUE" or "FALSE"	"TRUE"

Note:
Encryption is enabled as soon as one of these options is set.

An example to enable PDF encryption in Java code:

```
import org.apache.fop.pdf.PDFEncryptionParams;

[..]

FOUserAgent userAgent = fopFactory.newFOUserAgent();
useragent.getRendererOptions().put("encryption-params", new
PDFEncryptionParams(
    null, "password", false, false, true, true));
Fop fop = fopFactory.newFop(MimeConstants.MIME_PDF, userAgent);
[..]
```

The parameters for the constructor of PDFEncryptionParams are:

1. userPassword: String, may be null
2. ownerPassword: String, may be null
3. allowPrint: true if printing is allowed
4. allowCopyContent: true if copying content is allowed
5. allowEditContent: true if editing content is allowed
6. allowEditAnnotations: true if editing annotations is allowed

Alternatively, you can set each value separately in the Map provided by FOUserAgent.getRendererOptions() by using the following keys:

1. user-password: String
2. owner-password: String
3. noprint: Boolean or "true"/"false"
4. nocopy: Boolean or "true"/"false"
5. noedit: Boolean or "true"/"false"
6. noannotations: Boolean or "true"/"false"

4. Environment

In order to use PDF encryption, FOP has to be compiled with cryptography support. Currently, only JCE is supported. JCE is part of JDK 1.4. For earlier JDKs, it can be installed separately. The build process automatically detects JCE presence and installs PDF encryption support if possible, otherwise a stub is compiled in.

Cryptography support must also be present at run time. In particular, a provider for the RC4 cipher is needed. Unfortunately, the sample JCE provider in Sun's JDK 1.4 does **not** provide RC4. If you get a message saying

```
"Cannot find any provider supporting RC4"
```

then you don't have the needed infrastructure.

There are several commercial and a few Open Source packages which provide RC4. A pure Java implementation is produced by The Legion of the Bouncy Castle. Mozilla JSS is an interface to a native implementation.

5. Installing a crypto provider

The pure Java implementation from Bouncy Castle is easy to install.

1. Download the binary distribution for your JDK version. If you have JDK 1.3 or earlier you must also download a JCE from the same page.
2. Unpack the distribution. Add the jar file to your classpath. A convenient way to use the jar on Linux is to simply drop it into the FOP lib directory, it will be automatically picked up by `fop.sh`. If you have JDK 1.3 or earlier don't forget to install the JCE as well.
3. Open the `java.security` file and add `security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider`, preferably at the end of the block defining the other crypto providers. For JDK 1.4 this is detailed on Sun's web site.

If you have any experience with Mozilla JSS or any other cryptography provider, please post it to the fop-user list.