PDFlib GmbH

PLOP und PLOP DS

Version 5.2

Linearisierung, Optimierung, Sicherheit und Digitale Signaturen für PDF

Copyright © 1997–2017 PDFlib GmbH. Alle Rechte vorbehalten.

PDFlib GmbH
Franziska-Bilek-Weg 9, D-80339 München
www.pdflib.com
Tel. +49 • 89 • 452 33 84-0
Fax +49 • 89 • 452 33 84-99

Bei Fragen können Sie die PDFlib-Mailing-Liste abonnieren und sich deren Archiv ansehen unter: groups.yahoo.com/neo/groups/pdflib/info.

Vertriebsinformationen: sales@pdflib.com
Support für Inhaber einer kommerziellen PDFlib-Lizenz: support@pdflib.com (geben Sie bitte immer Ihre Lizenznummer an)

Der Inhalt dieser Dokumentation wurde mit größter Sorgfalt erstellt. PDFlib GmbH gibt jedoch keine Gewähr oder Garantie hinsichtlich der Richtigkeit oder Genauigkeit der Angaben in dieser Dokumentation und übernimmt keinerlei juristische Verantwortung oder Haftung für Schäden, die durch Fehler in dieser Dokumentation entstehen. Alle Warenbezeichnungen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen.

PDFlib und das PDFlib-Logo sind eingetragene Warenzeichen der PDFlib GmbH. PDFlib-Lizenznehmer sind dazu berechtigt, den Namen PDFlib und das PDFlib-Logo in ihrer Produktdokumentation zu verwenden. Dies ist jedoch nicht zwingend erforderlich.

Adobe, Acrobat, PostScript und XMP sind Warenzeichen von Adobe Systems Inc. AIX, IBM, OS/390, WebSphere, iSeries und zSeries sind Warenzeichen von International Business Machines Corporation. ActiveX, Microsoft, OpenType und Windows sind Warenzeichen von Microsoft Corporation. Apple, Macintosh und TrueType sind Warenzeichen von Apple Computer, Inc. Unicode und das Unicode-Logo sind Warenzeichen von Unicode, Inc. Unix ist ein Warenzeichen von The Open Group. Java und Solaris sind Warenzeichen von Sun Microsystems, Inc. HKS ist eine eingetragene Marke des HKS Warenzeichenverbands e.V.: Hostmann-Steinberg, K+E Printing Inks, Schmincke. Die Namen von anderen Produkten und Diensten können Warenzeichen von Unternehmen oder Organisationen sein, die hier nicht angeführt sind.

PDFlib PLOP und PLOP DS enthalten modifizierte Bestandteile folgender Software anderer Hersteller:
Zlib Compression Library, Copyright © 1995-2012 Jean-loup Gailly und Mark Adler
Kryptografische Software von Eric Young, Copyright © 1995-1998 Eric Young (eay@cryptsoft.com)
Software des OpenSSL Project für das OpenSSL-Toolkit (www.openssl.org)
XML-Parser Expat, Copyright © 1998, 1999, 2000 Thai Open Source Software Center Ltd
ICU International Components for Unicode, Copyright © 1995-2012 International Business Machines Corporation und andere
Libcurl multiprotocol file transfer library, Copyright © 1996-2014, Daniel Stenberg (daniel@haxx.se)

PDFlib PLOP und PLOP DS enthalten den Message-Digest-Algorithmus MD5 von RSA Security, Inc.

Inhaltsverzeichnis

0 Erste Schritte mit PLOP und PLOP DS 7

0.1 Installation der Software 7
0.2 Aktivieren des PLOP/PLOP DS-Lizenzschlüssels 9
0.3 Roadmap für Dokumentation und Beispiele 12
0.4 Übersicht über PLOP und PLOP DS 14
0.5 Was ist neu in PLOP und PLOP DS? 16

1 Funktionsumfang von PLOP 17

1.1 Kennwortschutz und Berechtigungen 17
1.2 Zertifikatsicherheit 19
1.3 Web-optimiertes (linearisiertes) PDF 20
1.4 Optimierung (Reduzierung der Dateigröße) 21
1.5 Reparaturmodus für beschädigtes PDF 22
1.6 Abfrage von Dokumentinformationen 23
1.7 Einfügen und Auslesen von Dokument-Infofeldern 24
1.8 Einfügen, Auslesen oder Entfernen von XMP-Metadaten 25
1.9 Details zur PDF-Verarbeitung mit PLOP 27

2 Funktionsumfang von PLOP DS (Digital Signature) 31

2.1 Signaturfunktionen in PLOP DS 31
2.2 Evaluierung von PLOP und PLOP DS 33
2.3 Signieren von Dokumenten mit PLOP DS 33
2.4 Zertifizierungssignaturen 34
2.5 Zeitstempel 35
2.6 Signaturen für Langzeitvalidierung (LTV) 35
2.7 PAdES-Signaturen 36
2.8 Visualisierung digitaler Signaturen 36
2.9 Abfrage von Signatureigenschaften 36

3 PLOP- und PLOP DS-Kommandozeilen-Tool 39

3.1 PLOP- und PLOP DS-Kommandozeilen-Optionen 39
3.2 Beispiele für PLOP- und PLOP DS-Kommandozeilen 43

4 Sprachbindungen der PLOP- und PLOP DS-Bibliothek 45

4.1 C-Sprachbindung 45
4.2 C++-Sprachbindung 48

4.3 COM-Sprachbindung 50
4.4 Java-Sprachbindung 51
4.5 .NET-Sprachbindung 53
4.6 Objective-C-Sprachbindung 54
4.7 Perl-Sprachbindung 56
4.8 PHP-Sprachbindung 57
4.9 Python-Sprachbindung 59
4.10 Ruby-Sprachbindung 60

5 Kennwortschutz 61

5.1 Kennwortschutz in PDF 61
5.2 Kennwortschutz für Dokumente mit PLOP einrichten 65
5.3 Anwenden von Kennwortschutz auf der Kommandozeile 68

6 Zertifikatsicherheit 71

6.1 Zertifikatsicherheit in Acrobat 71
6.2 Zertifikatsicherheit in PDF 75
6.2.1 CMS Enveloped Data 75
6.2.2 Kryptografische Details 77
6.3 Anwendungsfälle für Zertifikatsicherheit 80
6.4 Zertifikatsicherheit mit PLOP 82
6.5 Anwenden von Zertifikatsicherheit auf der Kommandozeile 86

7 Digitale Signaturen mit PLOP DS 89

7.1 Einführung 89
7.1.1 Grundbegriffe der digitalen Signatur 89
7.1.2 Signaturen in Acrobat und PDF 91
7.1.3 Vertrauenswürdige Stammzertifikate in Acrobat (Trusted Root Certificates) 93
7.2 Signieren von Dokumenten mit PLOP DS 96
7.2.1 Überblick 96
7.2.2 Signieren mit der integrierten Engine 97
7.2.3 PKCS#11-Engine für einen kryptografischen Token 97
7.2.4 PKCS#11-Engine für ein Hardware-Security-Modul (HSM) 99
7.2.5 Signieren mit der MSCAPI-Engine unter Windows 100
7.2.6 Kryptografische Details 102
7.3 PDF-Aspekte von Signaturen 105
7.3.1 Visualisieren von Signaturen mit Grafik oder Logo 105
7.3.2 Konformität zu PDF/A, PDF/UA, PDF/X und PDF/VT 107
7.3.3 Document Security Store (DSS) 110
7.3.4 Signaturen und inkrementelle PDF-Updates 111
7.3.5 Kombination von Verschlüsselung und Signaturen 112
7.3.6 Zertifizierungssignaturen 113

7.4 Sperrinformationen zu Zertifikaten 117
7.4.1 Online Certificate Status Protocol (OCSP) 117
7.4.2 Zertifikatsperrlisten (Certificate Revocation Lists) 120
7.4.3 OCSP oder CRL? 122

7.5 Zeitstempel 123
7.5.1 Konfiguration von Zeitstempeln 123
7.5.2 Signaturen mit eingebettetem Zeitstempel 124
7.5.3 Zeitstempelsignaturen auf Dokumentebene 125
7.5.4 Fehlerbehebung und nicht unterstütze TSAs 127

7.6 Langzeitvalidierung (LTV) 129
7.6.1 Langzeitvalidierung und Acrobat 129
7.6.2 LTV-fähige Signaturen in PLOP DS 130

7.7 Die Signaturstandards CAdES und PAdES 134
7.7.1 CMS- und CAdES-Signaturen 134
7.7.2 PAdES-Signaturen mit PLOP DS 137

8 API-Referenz für die PLOP- und PLOP DS-Bibliothek 139

8.1 Optionslisten 139

8.2 Allgemeine Funktionen 142

8.3 Funktionen für die Eingabe 145

8.4 Funktionen für die Ausgabe 149

8.5 Zertifikatsicherheit 154

8.6 Digitale Signaturen 156

8.7 Verarbeitung von Exceptions 167

8.8 Globale Optionen 169

8.9 Logging (Protokollierung) 171

8.10 pCOS-Funktionen 173

8.11 Funktion zur Unicode-Konvertierung 176

A Arbeiten mit Zertifikaten 179

B Kombination von PDFlib mit PLOP DS 181

C Kurzreferenz für die PLOP-Bibliothek 183

D Änderungen 185

0 Erste Schritte mit PLOP und PLOP DS

0.1 Installation der Software

PLOP und PLOP DS werden als kombiniertes Installationspaket für Windows ausgeliefert und als kombiniertes komprimiertes Archiv für alle anderen unterstützten Betriebssysteme. Installationspaket und Archiv enthalten das PLOP/PLOP DS-Kommandozeilen-Tool, die PLOP/PLOP DS-Bibliothek sowie Dokumentation und Beispiele. Nach dem Entpacken und Installieren des Pakets werden folgende Schritte empfohlen:

- Eine Einführung in die Funktionalität von PLOP und PLOP DS finden Sie in Kapitel 1, »Funktionsumfang von PLOP«, Seite 17 und Kapitel 2, »Funktionsumfang von PLOP DS (Digital Signature)«, Seite 31.
- Das PLOP/PLOP DS-Kommandozeilen-Tool kann sofort ausgeführt werden. Die unterstützten Optionen werden in Abschnitt 3.1, »PLOP- und PLOP DS-Kommandozeilen-Optionen«, Seite 39 beschrieben und werden außerdem angezeigt, wenn Sie das Tool ohne Optionen starten.
- Benutzern der PLOP/PLOP DS-Bibliothek/-Komponente empfehlen wir, die zu ihrer jeweiligen Umgebung passenden Abschnitte in Kapitel 4, »Sprachbindungen der PLOP- und PLOP DS-Bibliothek«, Seite 45 zu lesen und die installierten Beispiele durchzugehen. Unter Windows lassen sich die PLOP/PLOP DS-Programmierbeispiele für .NET über das Startmenü aufrufen oder bei anderen Sprachbindungen über das Installationsverzeichnis.

Wenn Sie eine kommerzielle PLOP/PLOP DS-Lizenz erworben haben, müssen Sie den Lizenzschlüssel eingeben, wie auf der nächsten Seite beschrieben.

Einschränkungen der Evaluierungsversion. Das PLOP/PLOP DS-Kommandozeilen-Tool sowie die Bibliothek können auch ohne kommerzielle Lizenz als voll funktionsfähige Evaluierungsversionen verwendet werden. Nicht lizenzierte PLOP/PLOP DS-Versionen dürfen nicht im produktiven Einsatz, sondern nur für die Evaluierung des Produkts verwendet werden. Zum produktiven Einsatz der Software benötigen Sie eine gültige Lizenz.

Nicht lizenzierte Versionen fügen den Text *unlicensed* in die Metadaten des Ausgabedokuments ein und stellen eine zusätzliche Titelseite an den Anfang des Dokuments. Zu Testzwecken lässt sich die zusätzliche Titelseite unterdrücken, wenn eine der folgenden Bedingungen erfüllt ist:

- Verschlüsselung mit dem vorgegebenen Kennwort *demo* bzw. *DEMO* (Optionen *userpassword* und *masterpassword)*.
- Anwenden einer Signatur mit einer digitalen ID, deren *Common Name (CN)* im Feld *subject* den Eintrag *demo* bzw. *DEMO* enthält; entsprechende digitale IDs sind in allen PLOP DS-Paketen zu Testzwecken enthalten.
- Verschlüsseln eines Dokuments mit Zertifikatsicherheit für Empfängerzertifikate, deren *Common Name (CN)* im Feld *subject* den Eintrag *demo* bzw. *DEMO* enthält; entsprechende Zertifikate sind in allen PLOP DS-Paketen zu Testzwecken enthalten.

Durch die zusätzliche Titelseite kann die PDF-Ausgabe in manchen Fällen nicht mehr PDF/A-, PDF/UA-, PDF/VT- oder PDF/X-konform sein, auch wenn die Eingabe einem dieser Standards entspricht. Die Nicht-Konformität bezieht sich dabei nur auf die Titelseite; sobald ein gültiger Lizenzschlüssel angewendet wird, besteht dieses Problem nicht mehr.

pCOS-Funktionen verarbeiten in der Evaluierungsversion nur PDF-Dokumente von maximal 10 Seiten und 1 MB Größe.

In der Evaluierungsversion ist für jedes von *plop.open_document( )* zurückgegebene Handle nur ein einziger Aufruf von *plop.create_document( )* erlaubt.

0.2 Aktivieren des PLOP/PLOP DS-Lizenzschlüssels

Zum produktiven Einsatz von PLOP/PLOP DS benötigen Sie einen gültigen Lizenzschlüssel. Nachdem Sie einen Lizenzschlüssel erworben haben, müssen Sie ihn anwenden, um die überflüssige Titelseite unterdrücken und beliebige Kennwörter verwenden zu können. Verwenden Sie zur Eingabe des Lizenzschlüssels eine der folgenden Methoden.

Wenn Sie die Option *frontpage* von *PLOP_set_option( )* auf *false* setzen, wird keine Titelseite am Dokumentanfang eingefügt, sondern eine Exception ausgelöst, wenn kein gültiger Lizenzschlüssel gefunden wurde.

Hinweis *PLOP/PLOP DS-Lizenzschlüssel sind plattformabhängig und können nur auf der Plattform eingesetzt werden, für die sie erworben wurden. Ein Lizenzschlüssel für PLOP DS aktiviert alle Funktionen von PLOP. Ein Lizenzschlüssel für PLOP hingegen kann die Signaturfunktionen nicht aktivieren, da diese nur in PLOP DS verfügbar sind.*

Windows-Installationsroutine. Windows-Benutzer können den Lizenzschlüssel bei der Installation von PLOP/PLOP DS in der mitgelieferten Installationsroutine angeben. Dies ist unter Windows empfehlenswert. Wenn Sie keine Schreibberechtigung für die Registry besitzen oder die Installationsroutine nicht verwenden können, müssen Sie auf eine der folgenden Methoden zurückgreifen.

Anwenden eines Lizenzschlüssels mit einem API-Aufruf zur Laufzeit. Fügen Sie in Ihrem Skript oder Programm eine Zeile ein, die den Lizenzschlüssel zur Laufzeit setzt. Der Parameter *license* muss dabei unmittelbar nach der Instantiierung des PLOP-Objekts gesetzt werden (d.h. nach *PLOP_new( )* oder einem ähnlichen Aufruf). Die Syntax hängt von der jeweiligen Programmiersprache ab:

- In COM/VBScript:

```
oPLOP.set_option "license=...Ihr Lizenzschlüssel..."
```

- In C++, Java, .NET/C#, Python und Ruby:

```
plop.set_option("license=...Ihr Lizenzschlüssel...")
```

- In C:

```
PLOP_set_option(p, "license=...Ihr Lizenzschlüssel...");
```

- In Perl und PHP:

```
$plop->set_option("license=...Ihr Lizenzschlüssel...")
```

Verwendung einer Lizenzdatei. Statt den Lizenzschlüssel mit einem Aufruf zur Laufzeit zu übergeben, können Sie ihn folgendermaßen in eine Textdatei eintragen (alle PLOP-Distributionen enthalten dafür die Vorlage *licensekeys.txt*). Mit einem '#'-Zeichen beginnende Zeilen enthalten Kommentare und werden ignoriert. Die zweite Zeile enthält Informationen zur Version der Lizenzdatei selbst:

```
# Lizenz-Information für Produkte der PDFlib GmbH
PDFlib license file 1.0
PLOP        5.2 ...Ihr Lizenzschlüssel...
```

Sie können Lizenzschlüssel für verschiedene Produkte der PDFlib GmbH in die Lizenzdatei aufnehmen, wobei jeder Schlüssel in einer eigenen Zeile stehen muss. Sie können

auch Lizenzschlüssel für verschiedene Plattformen aufnehmen, so dass die Lizenzdatei für mehrere Plattformen gemeinsam benutzt werden kann. Sie können Lizenzdateien folgendermaßen konfigurieren:

- Eine Datei namens *licensekeys.txt* wird an allen vorgegebenen Stellen gesucht (siehe »Standard-Dateisuchpfade«, Seite 10).
- Sie können den Parameter *licensefile* mit der API-Funktion *set_option( )* angeben:

  ```
  plop.set_option("licensefile={/Pfad/zu/Ihren/Lizenzschlüsseln.txt}");
  ```

- Übergeben Sie die Option *--plopopt* des PLOP-Kommandozeilen-Tools und die Option *licensefile* mit dem Namen einer Lizenzdatei:

  ```
  plop --plopopt "licensefile /Pfad/zu/Ihren/Lizenzschlüsseln.txt" ...
  ```

  Wenn der Pfadname Leerzeichen enthält, müssen Sie den Pfad mit Klammern umschließen:

  ```
  plop --plopopt "licensefile {/Pfad/zu/Ihrer Lizenzdatei.txt}" ...
  ```

- Sie können eine Umgebungsvariable anlegen, die auf die Lizenzdatei verweist. Unter Windows öffnen Sie die Systemsteuerung und wählen *System, Erweiterte Systemeinstellungen, Erweitert, Umgebungsvariablen*. Unter Unix verwenden Sie folgenden Befehl:

  ```
  export PDFLIBLICENSEFILE=/Pfad/zu/Ihren/Lizenzschlüsseln.txt
  ```

Lizenzschlüssel in der Registry. Unter Windows können Sie den Namen der Lizenzdatei auch in den folgenden Registry-Wert eintragen:

```
HKLM\SOFTWARE\PDFlib\PDFLIBLICENSEFILE
```

Alternativ können Sie den Lizenzschlüssel auch direkt in einen der folgenden Registry-Werte eintragen:

```
HKLM\SOFTWARE\PDFlib\PLOP5\license
HKLM\SOFTWARE\PDFlib\PLOP5\5.2\license
```

Der MSI-Installer schreibt den Lizenzschlüssel in den letzten dieser Einträge.

Hinweis *Seien Sie vorsichtig beim manuellen Zugriff auf die Registry von 64-Bit-Windows-Systemen: wie üblich funktionieren 64-Bit PLOP-Binärdateien mit der 64-Bit-Ansicht der Windows-Registry, während 32-Bit PDFlib-Binärdateien auf einem 64-Bit-System mit der 32-Bit-Ansicht der Registry funktionieren. Wenn Sie Registry-Werte für ein 32-Bit-Produkt manuell eintragen müssen, achten Sie darauf, die 32-Bit-Version des Werkzeugs* `regedit` *zu verwenden. Sie können es folgendermaßen über das Startmenü aufrufen:*

```
%systemroot%\syswow64\regedit
```

Standard-Dateisuchpfade. Unter Unix, Linux und OS X werden per Voreinstellung einige Verzeichnisse standardmäßig sogar ohne Angabe von Pfad und Verzeichnisnamen nach Dateien durchsucht. Die folgenden Verzeichnisse werden durchsucht:

```
<rootpath>/PDFlib/PLOP/5.2/resource/cmap
<rootpath>/PDFlib/PLOP/5.2/resource/codelist
<rootpath>/PDFlib/PLOP/5.2/resource/glyphlst
<rootpath>/PDFlib/PLOP/5.2/resource/fonts
```

```
<rootpath>/PDFlib/PLOP/5.2/resource/icc
<rootpath>/PDFlib/PLOP/5.2
<rootpath>/PDFlib/PLOP
<rootpath>/PDFlib
```

Unter Unix, Linux und OS X wird *<rootpath>* zuerst durch */usr/local* und dann durch das HOME-Verzeichnis ersetzt.

Voreingestellte Dateinamen für Lizenzdateien. Standardmäßig wird der folgende Dateiname in den Standard-Verzeichnissuchpfaden gesucht:

```
licensekeys.txt
```

Mit dieser Funktion lässt sich eine Lizenzdatei ohne die Angabe einer Umgebungsvariablen oder Laufzeit-Option verwenden.

Lizenzvarianten. Es gibt verschiedene Lizenzierungsmöglichkeiten für die Verwendung von PLOP auf einem oder mehreren Servern und für die Weitergabe von PLOP in eigenen Produkten. Wir bieten außerdem Support- und Wartungsverträge an. Bitte wenden Sie sich an uns, wenn Sie Fragen haben oder eine kommerzielle PLOP-Lizenz beziehen möchten:

PDFlib GmbH, Lizenzabteilung
Franziska-Bilek-Weg 9, D-80339 München
www.pdflib.com
Telefon+49 • 89 • 452 33 84-0
Fax+49 • 89 • 452 33 84-99
Vertrieb: *sales@pdflib.com*
Support für PDFlib-Lizenznehmer: *support@pdflib.com*

0.3 Roadmap für Dokumentation und Beispiele

Minibeispiele für PLOP. Die PLOP-Distribution enthält für jede unterstützte Sprachbindung eine Reihe von einfachen Programmbeispielen. Diese zeigen, wie Sie die PLOP-Bibliothek für elementare Programmieraufgaben nutzen:

- Das Beispiel *encrypt* verschlüsselt ein unverschlüsseltes PDF-Dokument mit Benutzer- und Master-Kennwort.
- Das Beispiel *certsec* verschlüsselt ein PDF-Dokument für ein oder mehrere Empfängerzertifikate. Beispiel-Empfängerzertifikate sind im PLOP-Paket enthalten. Das Paket enthält auch die entsprechenden digitalen IDs, die zur Entschlüsselung der verschlüsselten Dokumente mit PLOP oder zum Öffnen in Acrobat benötigt werden. Das Kennwort für alle Dateien mit digitalen IDs (z.B. *demo_recipient_1.p12)* lautet *demo*.
- Das Beispiel *dumper* nutzt die pCOS-Schnittstelle, um allgemeine Eigenschaften und Informationen über den Verschlüsselungs- und Signaturstatus eines Dokuments sowie Dokumentinformationen und XMP-Metadaten abzufragen.
- Das Beispiel *insertxmp* liest XMP-Metadaten aus einer Datei und fügt das XMP in ein PDF-Dokument ein. XMP-Beispieldateien sind in allen PLOP-Paketen zu Testzwecken enthalten.

Minibeispiele für PLOP DS. Die folgenden Minibeispiele zeigen den Einsatz von PLOP DS:

- Das Beispiel *sign* zeigt, wie Sie ein vorhandenes PDF-Dokument mit einer digitalen Signatur versehen.
- Das Beispiel *multisign* zeigt, wie Sie mehrere PDF-Dokumente mit einer digitalen Signatur versehen und demonstriert Session-Handling für PKCS#11-Tokens.
- Das Beispiel *hellosign* zeigt, wie Sie ein Dokument dynamisch mit PDFlib erstellen und es im Speicher an PLOP DS übergeben. PLOP DS versieht das Dokument dann mit einer digitalen Signatur. Für dieses Beispiel benötigen Sie das Produkt PDFlib, das nicht im PLOP-Paket enthalten ist. Auf unserer Webseite finden Sie bei Bedarf kostenlose Evaluierungsversionen von PDFlib.
- Das Beispiel *dynamicsign* zeigt, wie Sie ein Visualisierungsdokument für die Signatur mit PDFlib dynamisch erzeugen, z.B. mit einem personalisierten Bild einer handgeschriebenen Unterschrift. Für diese Beispiele ist das Produkt PDFlib ebenfalls erforderlich.

Die Signaturbeispiele sind für die Verwendung digitaler Demo-IDs vorbereitet, die im Paket enthalten sind. Das Kennwort für alle Dateien mit digitalen IDs (wie z.B. *demo_signer_rsa_2048.p12)* lautet *demo*.

PLOP-Kommandozeilenbeispiele. Das PLOP-Kommandozeilen-Tool unterstützt verschiedene Optionen, die in Abschnitt 3.1, »PLOP- und PLOP DS-Kommandozeilen-Optionen«, Seite 39 beschrieben werden. Weitere Aufrufe des PLOP-Kommandozeilen-Tools finden Sie in diversen anderen Kapiteln dieses Handbuchs.

pCOS Cookbook. Das *pCOS Cookbook* ist eine Sammlung von Codefragmenten für die pCOS-Schnittstelle, die in PLOP und PLOP DS enthalten ist. Es steht unter folgender Adresse zur Verfügung: *www.pdflib.com/pcos-cookbook*.

Die pCOS-Schnittstelle wird in der pCOS-Pfadreferenz beschrieben, die im PLOP-Paket enthalten ist.

0.4 Übersicht über PLOP und PLOP DS

PLOP ist in zwei Varianten verfügbar: PLOP stellt das Basisprodukt und PLOP DS die erweiterte Variante dar.

Funktionsumfang von PLOP. PLOP bietet folgende Verarbeitungsmöglichkeiten für PDF-Dokumente:

- Kennwortschutz: PDF-Dokument mit Benutzer- und/oder Master-Kennwort verschlüsseln; PDF-Verschlüsselung entfernen, sofern das Master-Kennwort des Dokuments bekannt ist; Berechtigungen wie »Drucken nicht zulässig« oder »Textextraktion nicht zulässig« hinzufügen oder entfernen, sofern das Master-Kennwort des Dokuments bekannt ist.
- Zertifikatsicherheit: PDF-Dokument für ein oder mehrere Empfängerzertifikate verschlüsseln; geschütztes PDF-Dokument mit einer passenden digitalen ID entschlüsseln. Berechtigungseinstellungen für Zertifikatsicherheit anwenden oder entfernen.
- Linearisierte PDF-Dokumente zur schnelleren Übertragung vom Webserver erzeugen.
- Größe von PDF-Dokumenten durch Entfernen unnötiger Objekte optimieren.
- Beschädigte PDF-Dokumente reparieren.
- Mit der integrierten Programmierschnittstelle pCOS verschiedenste Informationen über die Sicherheitseigenschaften des Dokuments (verschlüsselt mit Benutzer- oder Master-Kennwort) abfragen, zum Beispiel Berechtigungseinstellungen, Dokument-Metadaten usw.
- Vordefinierte und benutzerdefinierte Dokument-Infofelder abfragen und setzen.
- XMP-Metadaten einfügen und extrahieren.

Funktionsumfang von PLOP DS. PLOP DS (Digital Signature) enthält alle Funktionen von PLOP und bietet darüber hinaus die Möglichkeit, PDF-Dokumente digital zu signieren. Die Signaturen unterstützen Zeitstempel, Langzeitvalidierung *(Long-Term Validation, LTV)* und PAdES-Signaturen. Abschnitt 2.1, »Signaturfunktionen in PLOP DS«, Seite 31 gibt einen Überblick über die Funktionalität für digitale Signaturen in PLOP DS.

Vorteile. PDFlib PLOP und PLOP DS bieten folgende Vorteile:

- PLOP/PLOP DS berücksichtigt die Standards PDF/A, PDF/UA, PDF/VT und PDF/X: entspricht das Eingabedokument einem dieser Standards, so ist dies auch für die Ausgabe gewährleistet, soweit möglich. Bewirkt eine Operation (z.B. Verschlüsselung einer PDF/A-Eingabedatei) eine Ausgabe, die nicht Standard-konform ist, so wird die Operation abgelehnt oder die Standardidentifikation entfernt.
- PLOP/PLOP DS ist ein eigenständiges Produkt, das keine Fremdsoftware benötigt, um PDF-Dokumente einzulesen, zu verschlüsseln, zu signieren oder auszugeben.
- PLOP/PLOP DS ist für den Servereinsatz optimiert (thread-sicher und frei von Speicherlecks). PLOP erfüllt alle Qualitäts- und Geschwindigkeitsanforderungen für den Einsatz auf großen Servern und kann auf dem Webserver, für umfangreiche Batch-Prozesse usw. verwendet werden.
- PLOP/PLOP DS wird für zahlreiche Plattformen und verschiedenste Programmierumgebungen angeboten.
- PLOP/PLOP DS ist flexibel und sowohl als Kommandozeilen-Tool als auch als Software-Bibliothek (Komponente) für verschiedene Entwicklungsumgebungen verfügbar.

PLOP/PLOP DS als Kommandozeilen-Tool oder als Bibliothek? PLOP/PLOP DS wird als Software-Bibliothek (Komponente) für verschiedene Entwicklungsumgebungen sowie als Kommandozeilen-Tool für Batch-Prozesse ausgeliefert. Beide bieten den gleichen Funktionsumfang, eignen sich aber für unterschiedliche Einsatzbereiche. Im folgenden finden Sie einige Anhaltspunkte für die Wahl der geeigneten Variante:

- Das PLOP/PLOP DS-Kommandozeilen-Tool eignet sich für die Stapelverarbeitung von PDF-Dokumenten. Es erfordert keine Programmierung, sondern kann über leistungsfähige Kommandozeilen-Optionen gesteuert und damit in komplexe Arbeitsabläufe integriert werden. Sie können das PLOP/PLOP-DS-Kommandozeilen-Tool auch in Umgebungen verwenden, die den Einsatz der PLOP/PLOP-DS-Bibliothek nicht unterstützen.
- Die PLOP/PLOP DS-Software-Bibliothek lässt sich in zahlreiche Programmierumgebungen einbinden, etwa in .NET, Java (inklusive Servlets), PHP sowie reine C- oder C++-Anwendungsentwicklung.

Die PLOP/PLOP DS-Lizenz beinhaltet sowohl das Kommandozeilen-Tool als auch die Software-Bibliothek.

0.5 Was ist neu in PLOP und PLOP DS?

Änderungen in PLOP 5.1:

- Zertifikatsicherheit: Verschlüsseln eines Dokuments für eine Gruppe von Empfängern, die durch ihr digitales Zertifikat identifiziert werden
- pCOS-Interface 11 zur Abfrage von Informationen über Dokumente mit Zertifikatsicherheit
- aktualisierte Sprachbindungen und Plattform-Unterstützung
- zahlreiche Fehlerbehebungen und Verbesserungen bei den Sprachbindungen sowie am PLOP-Kern
- (PLOP 5.2) Zertifikatsicherheit: Unterstützung des OAEP-Padding-Schemas für RSA

Signaturbezogene Änderungen in PLOP DS 5.1:

- aktualisierte Zeitstempel gemäß RFC 5816 *(SigningCertificateV2/ESSCertIDv2)*
- Optimierung von Dateigröße und Verarbeitungszeit beim Erzeugen von Signaturen
- Optionen für Zeittoleranzen bei OCSP
- Unterstützung für indirekte CRLs
- Robustere Verarbeitung von CRL-Anfragen, z.B. unerwartete HTTP-Header
- Workaround für das Verhalten bestimmter Token-Modelle in der PKCS#11-Engine
- Verbesserungen bei PKCS#11 für Multi-Threaded-Signieren
- Unterstützung für das Signieren mit einem Hardware-Security-Modul (HSM)
- Standardmäßige Erstellung von PAdES-/CAdES-Signaturen
- benutzerdefinierter Build für den Anschluss einer externen Krypto-Engine zum Hashing und Signieren
- Fehlerbehebung bei der PDF-Verarbeitung, z.B. bei Formularfeldnamen, XMP-Properties
- Codebeispiele für die Erzeugung dynamischer Visualisierungsdokumente mit PDFlib
- neue Build-Konfiguration für die Anbindung externer Krypto-Routinen über die PKCS#11-Schnittstelle ohne dynamisches Laden
- (PLOP 5.2) Unterstützung des PSS-Encoding-Schemas für RSA

1 Funktionsumfang von PLOP

Hinweis *Zur PLOP DS-Funktionalität für digitale Signaturen siehe Kapitel 2, »Funktionsumfang von PLOP DS (Digital Signature)«, Seite 31.*

1.1 Kennwortschutz und Berechtigungen

Verschlüsselung und Entschlüsselung von PDF-Dokumenten mit Kennwörtern sowie Berechtigungseinstellungen werden ausführlich in Kapitel 5, »Kennwortschutz«, Seite 61 behandelt. Der vorliegende Abschnitt gibt lediglich einen kurzen Überblick sowie einige einführende Beispiele.

Abfrage von Sicherheitseinstellungen. Mit der Programmierschnittstelle pCOS können Sie verschiedene Sicherheitseigenschaften eines mit Kennwortschutz versehenen PDF-Dokuments abfragen. Die dazu erforderlichen Funktionsaufrufe und -parameter finden Sie im Minibeispiel *dumper,* das in allen PLOP-Paketen enthalten ist. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--info* (ein Beispiel hierzu finden Sie in Abschnitt 1.6, »Abfrage von Dokumentinformationen«, Seite 23).

Verschlüsselung von Dokumenten mit Kennwort. Mit den Optionen *userpassword* oder *masterpassword* (oder beiden) von *PLOP_create_document( )* können Sie Dokumente verschlüsseln. Beachten Sie dabei, dass ein Benutzerkennwort immer auch ein Master-Kennwort erfordert, aber nicht umgekehrt. Ein Codebeispiel zur Verschlüsselung von PDF-Dokumenten finden Sie im Programmbeispiel *encrypt,* das in allen PLOP-Paketen enthalten ist. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Optionen *--user* und *--master*.

Beispiel: Verschlüsseln einer Datei mit Benutzerkennwort *demo* und Master-Kennwort *DEMO:*

```
plop --user demo --master DEMO --outfile encrypted.pdf input.pdf
```

Festlegen von Berechtigungseinstellungen. Zur Festlegung von Berechtigungseinstellungen verwenden Sie die Option *permissions* von *PLOP_create_document( ),* die verschiedene Schlüsselwörter unterstützt (siehe Tabelle 5.3, Seite 66). Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--permissions*. Beachten Sie dabei, dass zum Ändern von Berechtigungseinstellungen immer ein Master-Kennwort erforderlich ist.

Beispiel: Dokument mit Master-Kennwort *DEMO* verschlüsseln und Drucken des Dokuments sowie Kopieren von Inhalten untersagen:

```
plop --master DEMO --permissions "noprint nocopy" --outfile encrypted.pdf input.pdf
```

Entschlüsselung kennwortgeschützter Dokumente. Zum Entschlüsseln eines Dokuments übergeben Sie das passende Benutzer- oder Master-Kennwort mit der Option *password* von *PLOP_open_document( )*. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--password*.

Beispiel: Entschlüsseln einer Datei mit dem Master-Kennwort *DEMO.* Alle im Eingabedokument eventuell gesetzten Zugriffsbeschränkungen werden dabei entfernt (da die Ausgabe nicht verschlüsselt ist):

```
plop --password DEMO --outfile decrypted.pdf encrypted.pdf
```

Für weitere Beispiele zur Ver- und Entschlüsselung siehe Abschnitt 5.3, »Anwenden von Kennwortschutz auf der Kommandozeile«, Seite 68.

1.2 Zertifikatsicherheit

Verschlüsselung und Entschlüsselung von PDF-Dokumenten mit Zertifikatsicherheit wird ausführlich in Kapitel 6, »Zertifikatsicherheit«, Seite 71 behandelt. Der vorliegende Abschnitt gibt lediglich einen kurzen Überblick sowie einige einführende Beispiele.

Abfrage von Sicherheitseinstellungen. Mit der Programmierschnittstelle pCOS können Sie verschiedene Sicherheitseigenschaften eines PDF-Dokuments mit Zertifikatsicherheit abfragen. Die dazu erforderlichen Funktionsaufrufe und -parameter finden Sie im Minibeispiel *dumper,* das in allen PLOP-Paketen enthalten ist. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--info* (ein Beispiel hierzu finden Sie in Abschnitt 1.6, »Abfrage von Dokumentinformationen«, Seite 23).

Verschlüsselung von Dokumenten mit Zertifikat. Zur Verschlüsselung eines Dokuments übergeben Sie das Empfängerzertifikat mit der Option *certificate* von *PLOP_add_recipient( )*. Ein Codebeispiel zur Verschlüsselung von PDF-Dokumenten finden Sie im Programmbeispiel *certsec,* das in allen PLOP-Paketen enthalten ist. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--recipient*.

Beispiel: Verschlüsseln einer Datei mit Zertifikat:

```
plop --recipient "certificate={filename=demo_recipient_1.pem}" ↵
        --outfile encrypted.pdf input.pdf
```

Festlegen von Berechtigungseinstellungen. Zur Festlegung von Berechtigungseinstellungen für einen Empfänger verwenden Sie die Option *permissions* von *PLOP_add_recipient( )*.

Beispiel: Verschlüsseln eines Dokuments für einen Empfänger und Einschränken seiner Berechtigungen, so dass Drucken und Kopieren nicht erlaubt sind:

```
plop --recipient "certificate={filename=demo_recipient_1.pem permissions={noprint ↵
        nocopy}}" --outfile encrypted.pdf input.pdf
```

Entschlüsselung von Dokumenten mit Zertifikatsicherheit. Zur Entschlüsselung eines Dokuments übergeben Sie die zugehörige digitale ID des Empfängers mit der Option *digitalid* von *PLOP_open_document( )*. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--inputopt*.

Beispiel: Entschlüsseln einer einzelnen Datei mit einer digitalen ID, die in einer kennwortgeschützten PKCS#12-Datei verfügbar ist:

```
plop --inputopt "digitalid={filename=demo_recipient_1.p12} password=demo" ↵
     --outfile decrypted.pdf encrypted.pdf
```

Für weitere Beispiele zur Ver- und Entschlüsselung siehe Abschnitt 6.5, »Anwenden von Zertifikatsicherheit auf der Kommandozeile«, Seite 86.

1.3 Web-optimiertes (linearisiertes) PDF

PLOP bietet die Möglichkeit, auf PDF-Dokumente die sogenannte Linearisierung anzuwenden. Die daraus resultierende Dokumenteigenschaft wird in Acrobat *Schnelle Webanzeige* genannt. Bei der Linearisierung werden die Objekte in der PDF-Datei umgeordnet und Informationen hinzugefügt, die dem schnelleren Zugriff dienen.

Während nicht linearisierte PDF-Dokumente vollständig zum Client übertragen werden müssen, kann ein Webserver linearisierte PDF-Dokumente seitenweise mit einem Verfahren namens Byteserving transferieren. Damit kann Acrobat (als Browser-Plugin) ein PDF-Dokument auch teilweise abrufen. Die erste Dokumentseite wird Benutzern dann unverzüglich angezeigt, ohne dass sie die vollständige Übertragung des Dokuments vom Server abwarten müssen. Dies erhöht die Benutzerfreundlichkeit.

Beachten Sie, dass nicht PLOP, sondern der Webserver die PDF-Daten an den Browser übergibt. PLOP bereitet die PDF-Dateien lediglich zum Byteserving vor. Zum Byteserving von PDF sind folgende Voraussetzungen zu erfüllen:

- Das PDF-Dokument muss linearisiert worden sein. Mit PLOP kann die Linearisierung gleichzeitig mit einer Ver- oder Entschlüsselung des Dokuments erfolgen. In Acrobat können Sie in den Dokumenteigenschaften nachsehen, ob eine Datei linearisiert ist (»Schnelle Webanzeige: Ja«).
- Acrobat muss als Browser-Plugin installiert sein und in Acrobat müssen Sie seitenweises Herunterladen im PDF-Viewer aktiviert haben (Acrobat X/XI/DC: *Bearbeiten, Voreinstellungen, [Allgemein] Internet, Schnelle Webanzeige zulassen)*. Diese Einstellung ist standardmäßig aktiviert.

Je größer eine PDF-Datei ist (in Seiten oder MB), desto mehr wird sie bei der Übertragung über das Web von der Linearisierung profitieren.

Die Linearisierung kann in Kombination mit Ver- und Entschlüsselung angewendet werden. Um eine geschützte Datei zu linearisieren, muss das passende Master-Kennwort übergeben werden (siehe Tabelle 5.2).

Linearisierung kleiner Dateien. Da Linearisierung für die Verbesserung der webbasierten Anzeige von großen PDF-Dokumenten gedacht ist, macht sie bei einseitigen Dokumenten nicht viel Sinn (obwohl dies möglich ist). Acrobat behandelt kleine linearisierte Dokumente jedoch nicht immer als linearisiert. In Acrobat gelten beispielsweise alle Dokumente kleiner als 4KB als nicht linearisiert.

Linearisierung von PDF-Dokumenten mit PLOP. Zur Linearisierung verwenden Sie die Option *linearize* von *PLOP_create_document( )*.

Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--webopt*. Beispiel: Alle PDF-Dokumente eines Verzeichnisses linearisieren (unter der Annahme, dass diese kein Kennwort benötigen) und die Ergebnisdokumente in das Verzeichnis *output* kopieren. Mit dem Verbosity-Level 2 werden die Namen aller Ein- und Ausgabedateien bei der Verarbeitung ausgegeben:

```
plop --verbose 2 --webopt --targetdir output *.pdf
```

1.4 Optimierung (Reduzierung der Dateigröße)

PLOP ist in der Lage, PDF-Dokumente bei der Verarbeitung zu optimieren:

- PLOP erkennt mehrfach vorkommende identische Daten und entfernt alle überflüssigen Informationen. Dies ist vorwiegend bei Fonts und Bildern relevant, kann sich aber auch auf andere Arten von Daten auswirken, z.B. auf ICC-Profile oder auf Seiten mit komplett identischem Inhalt. Ein eingebetteter Font bzw. ein eingebettetes Bild wird entfernt, wenn ein anderer Font bzw. ein anderes Bild aus genau denselben Daten besteht; alle Referenzen auf die entfernten Daten werden durch Referenzen auf die verbleibende Instanz des Fonts oder Bildes ersetzt. Setzt sich ein Dokument beispielsweise aus mehreren PDFs zusammen, die jeweils einen Teil des Dokuments enthalten, wobei jedes denselben Font eingebettet hat, enthält das zusammengesetzte Dokument unter Umständen redundante Fontinformationen. PLOP entfernt dann alle überflüssigen Fontinformationen, so dass nur eine einzige Instanz des Fonts erhalten bleibt.
- Unnötige Objekte werden aus der PDF-Datei durch einen Prozess namens *Garbage Collection* entfernt. In manchen Fällen fügt Acrobat Änderungen am Ende einer Datei an, so dass der vorige Dokumentzustand erhalten bleibt (zum Beispiel, wenn Sie die Datei in Acrobat mit *Speichern* statt mit *Speichern unter...* sichern). PLOP entfernt dann alle Objekte, die sich auf ältere Fassungen des Dokuments beziehen.

Keiner der von PLOP durchgeführten Optimierungsschritte führt zu Informationsverlust (z.B. Rücknahme der Fonteinbettung oder verringerte Bildauflösung). Alle Informationen, die für die Anzeige und den Druck des Dokuments in unveränderter Qualität relevant sind, bleiben erhalten.

Da nur eine geringe Anzahl heutiger PDF-Dokumente redundante Objekte enthält, ist der Optimierungsschritt standardmäßig deaktiviert.

Optimierung von PDF-Dokumenten mit PLOP. Der Optimierungsschritt lässt sich mit der Option *optimize=all* von *PLOP_create_document( )* oder der Option *--outputopt* des PLOP-Kommandozeilen-Tools aktivieren.

Beispiel: Optimierung eines Dokuments mit dem PLOP-Kommandozeilen-Tool:

```
plop --outputopt optimize=all --outfile optimized.pdf input.pdf
```

Entfernen von XMP-Metadaten mit PLOP. Manche Anwendungen erzeugen PDF-Ausgabe mit einer großen Menge an XMP-Metadaten, die nicht immer erforderlich sind. In Einzelfällen machen die XMP-Metadaten sogar den Großteil des Gesamtvolumens der PDF-Datei aus. In solchen Fällen können Sie mit PLOP unerwünschte XMP-Metadaten folgendermaßen entfernen:

```
plop --inputopt xmppolicy=remove --outfile output.pdf input.pdf
```

Damit lässt sich die Größe der PDF-Datei zu Lasten ausführlicher Metadaten erheblich verringern. Beachten Sie, dass Standard-Identifikatoren (z.B. für PDF/A) im XMP verloren gehen.

1.5 Reparaturmodus für beschädigtes PDF

In PLOP ist ein Reparaturmodus für beschädigte PDF-Dateien implementiert, so dass sich selbst manche beschädigten Dokumente verarbeiten lassen. Trotzdem kann es in seltenen Fällen vorkommen, dass PLOP ein beschädigtes Dokument zurückweist, weil es nicht repariert werden kann.

Reparatur von PDF-Dokumenten mit PLOP. Der Reparaturmodus wird automatisch aktiviert, sobald PLOP auf beschädigte Eingabe stößt. Mit der Option *repair=force* von *PLOP_open_document()* können Sie den Reparaturmodus erzwingen, selbst wenn beim Öffnen des PDF-Dokuments kein Fehler aufgetreten ist. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--inputopt repair=force*. Mit *repair=none* deaktivieren Sie den Reparaturmodus.

Beispiel: Unbedingte Rekonstruktion eines Dokuments mit dem PLOP-Kommandozeilen-Tool:

```
plop --inputopt repair=force --outfile repaired.pdf damaged.pdf
```

Ungültige XMP-Metadaten. Mit PLOP lassen sich bestimmte Probleme bei XMP-Metadaten reparieren. Trotzdem kann es vorkommen, dass dies nicht möglich ist. Durch XMP-Metadaten hervorgerufene Fehler beim Parsen von XMP führen immer zu ungültigem XMP. Bei ungültigem XMP lässt sich in PLOP mit der Option *xmppolicy* die Verarbeitung steuern. Für weitere Informationen siehe »Umgang mit ungültigen XMP-Metadaten«, Seite 26.

1.6 Abfrage von Dokumentinformationen

Ausführliche Informationen zur pCOS-Schnittstelle finden Sie in der pCOS-Pfadreferenz.

Mit der Programmierschnittstelle pCOS, die in die PLOP-Bibliothek integriert ist, lassen sich verschiedenste Eigenschaften eines PDF-Dokuments abfragen. Ein Codebeispiel zur Abfrage von Dokumentinformationen mit pCOS finden Sie im Minibeispiel *dumper,* das in allen PLOP-Paketen enthalten ist.

Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option --*info*.Beispiel: Anzeige von Sicherheits- und anderen Dokumenteigenschaften in einem PDF-Dokument:

```
plop --info *.pdf
```

Der Programmaufruf bewirkt in etwa folgende Ausgabe:

```
          File name: PLOP-manual.pdf
        PDF version: 1.7
         Encryption: No encryption
          Master pw: false
            User pw: false
             nocopy: false (copying is allowed)
           nomodify: false (adding form fields and other changes is allowed)
           noannots: false (adding or changing comments or form fields is allowed)
         noassemble: false (insert/delete/rotate pages, creating bookmarks is allowed)
            noforms: false (filling form fields is allowed)
      noaccessible: false (extracting text or graphics for accessibility is allowed)
      nohiresprint: false (high-resolution printing is allowed)
     plainmetadata: true  (metadata is not encrypted)
        Linearized: true
       PDF/X status: none
       PDF/A status: none
      PDF/UA-Status: none
      PDF/VT-Status: none
         Tagged PDF: false
         Signatures: 0
    Reader-enabled: false

       No. of pages: 172
       No. of fonts: 12
           embedded TrueType font PDFlibLogo-Regular
           embedded Type 1 CFF font ThesisAntiqua-Bold
           embedded Type 1 CFF font TheSans-Italic
           ...
           embedded Type 1 CFF font ThesisAntiqua-Normal
           embedded Type 1 CFF font TheSansMonoCondensed-Plain

             Author: 'PDFlib GmbH'
       CreationDate: 'D:20160420105759Z'
            Creator: 'FrameMaker 11.0.2'
            ModDate: 'D:20160420112723+02'00''
           Producer: 'Acrobat Distiller 11.0 (Windows)'
            Subject: 'PDFlib PLOP and PLOP DS: PDF Linearization, Optimization,
Protection, Digital Signature'
              Title: 'PDFlib PLOP and PLOP DS Manual'

      XMP meta data: is present
     Encr. attachm.: no
```

1.7 Einfügen und Auslesen von Dokument-Infofeldern

PDF unterstützt zwei Arten von Dokument-Metadaten, die allgemeine Informationen über ein Dokument enthalten: Dokument-Infofelder und XMP-Metadaten.

Dokument-Infofelder sind Schlüssel, denen Strings mit unstrukturierten Informationen zugeordnet sind. Üblicherweise verwendet werden die vordefinierten Infoschlüssel *Subject* (Thema), *Title* (Titel), *Author* (Verfasser) und *Keywords* (Stichwörter). Für besondere Zwecke können beliebige benutzerdefinierte Schlüssel festgelegt werden. Dokument-Infofelder stellen die einfache, herkömmliche Art von PDF-Metadaten dar.

Mit PLOP können Sie neue Dokument-Infofelder hinzufügen oder existierende mit anderen Inhalten versehen. Sowohl vordefinierte als auch benutzerdefinierte Einträge lassen sich setzen. Falls das Dokument XMP-Metadaten enthält, werden alle vordefinierten Dokument-Infofelder automatisch mit den XMP-Metadaten synchronisiert, um die Metadaten konsistent zu halten.

Einfügen von Dokument-Infofeldern mit PLOP. Um Dokument-Infofelder zu setzen, verwenden Sie die Option *docinfo* von *PLOP_create_document( )*.

Beispiel: Festlegen des vordefinierten Dokument-Infofelds *Subject* sowie des benutzerdefinierten Dokument-Infofelds *Department;* die Klammern um *Product Manual* schützen das Leerzeichen:

```
docinfo={Department Techdoc Subject {Product Manual}}
```

Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--outputopt* wie folgt:

```
plop --outputopt "docinfo={Department Techdoc Subject {Product Manual}}" ↵
        --outfile output.pdf input.pdf
```

Auslesen von Dokument-Infofeldern mit PLOP. Mit der Programmierschnittstelle pCOS, die in die PLOP-Bibliothek integriert ist, lassen sich Dokument-Infofelder (Schlüssel und Werte) eines PDF-Dokuments abfragen. Die dazu erforderlichen Funktionsaufrufe und -parameter finden Sie im Minibeispiel *dumper,* das in allen PLOP-Paketen enthalten ist.

Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--info* (ein Beispiel hierzu finden Sie in Abschnitt 1.6, »Abfrage von Dokumentinformationen«, Seite 23).

Dokument-Infofelder in PDF/A. Beachten Sie, dass der PDF/A-Standard eine besondere Behandlung von Dokument-Infofeldern erfordert:

- PDF/A-1: die vordefinierten Dokument-Infofelder *Title, Author, Subject, Keywords, Creator, Producer, CreationDate, ModDate* müssen mit den XMP-Metadaten des Dokuments synchronisiert werden. PLOP führt diese Synchronisation automatisch durch.
- PDF/A-2/3: Dokument-Infofelder können vorhanden sein, müssen aber von PDF/A-konformen Readern ignoriert werden. Sind Dokument-Infofelder vorhanden, sollten sie mit den XMP-Metadaten des Dokuments synchronisiert werden, was von PLOP wie bei PDF/A-1 automatisch durchgeführt wird.

1.8 Einfügen, Auslesen oder Entfernen von XMP-Metadaten

XMP *(Extensible Metadata Platform)* ist ein XML-Framework mit zahlreichen vordefinierten Properties. Wie auch der Name besagt, kann XMP durch benutzerdefinierte Extension-Schemas erweitert werden, um speziellen Anforderungen zu genügen. XMP ist weitaus leistungsfähiger als Dokument-Infofelder und wird zudem vom Standards wie z.B. PDF/A gefordert. Verschiedene Branchenverbände haben Empfehlungen für den Einsatz von XMP in vertikalen Anwendungen veröffentlicht, so z.B. für digitale Bildverarbeitung oder für den Datenaustausch in der Druckvorstufe.

Detaillierte Informationen zu XMP sowie Links zu anderen Ressourcen finden Sie unter *www.pdflib.com/knowledge-base/xmp-metadata*.

Mit PLOP können Sie XMP-Metadaten in PDF-Dokumente einfügen und XMP daraus auslesen. Die eingefügten XMP-Daten werden validiert, um sicherzustellen, dass die generierte Ausgabe gültig ist. Genügt das Eingabedokument dem PDF/A-Standard, müssen die vom Benutzer übergebenen XMP-Daten die in PDF/A festgelegten XMP-Regeln einhalten. PLOP überprüft die Daten hinsichtlich dieser Regeln (einschließlich der Überprüfung von XMP-Extension-Schemas), damit garantiert ist, dass die PDF/A-Eingabe mit den übergebenen XMP-Daten in konforme PDF/A-Ausgabe überführt wird.

Das Einfügen von XMP mit PLOP kann in folgenden und vielen weiteren Fällen verwendet werden (die Namen der XMP-Beispieldateien in der PLOP-Distribution sind jeweils in Klammern angegeben):

- XMP-Metadaten in PDF/A-Dokumente einfügen; XMP-Extension-Schemas für PDF/A werden dabei unterstützt *(machine_pdfa1.xmp)*.
- XMP-Metadaten einfügen, die den Scanprozess für digitalisierte Akten beschreiben *(engineering.xmp)*.
- XMP-Metadaten gemäß Ad-Ticket-Schema der Ghent Workgroup (GWG) einfügen *(gwg_ad_ticket.xmp)*. Für weitere Informationen siehe *www.gwg.org/download/jobtickets/*
- Firmenspezifische XMP-Metadaten einfügen *(acme.xmp)*.

Einfügen von XMP-Metadaten mit PLOP. Zum Einfügen von Metadaten müssen Sie eine Datei erstellen, die gültige XMP-Metadaten im Format UTF-8 enthält. Zum Einfügen von XMP verwenden Sie die Option *metadata* von *PLOP_create_document( )*, die mehrere Unteroptionen unterstützt. Ein Codebeispiel zum Einfügen von XMP in PDF-Dokumente finden Sie im Minibeispiel *insertxmp*, das in allen PLOP-Paketen enthalten ist.

Beispiel: Einfügen von XMP-Metadaten aus einer Datei namens *gwg_ad_ticket.xmp*, wobei die XMP-Daten hinsichtlich des Standards XMP 2004 validiert werden:

```
plop --outputopt "metadata={filename=gwg_ad_ticket.xmp validate=xmp2004}" ↵
        --outfile output.pdf input.pdf
```

Auslesen von XMP-Metadaten mit PLOP. Mit der Programmierschnittstelle pCOS, die in die PLOP-Bibliothek integriert ist, lassen sich XMP-Metadaten eines PDF-Dokuments auslesen. Die dazu erforderlichen Funktionsaufrufe und -parameter finden Sie im Minibeispiel *dumper*, das in allen PLOP-Paketen enthalten ist. Das Codebeispiel im Programm *dumper* gibt allerdings nicht die XMP-Metadaten selbst, sondern lediglich die Größe der im Dokument gefundenen XMP-Daten aus.

Das PLOP-Kommandozeilen-Tool können Sie zur Extraktion von XMP-Metadaten nicht verwenden. Hierzu bietet PDFlib das leistungsfähige pCOS-Kommandozeilen-Tool an.

Entfernen von XMP-Metadaten mit PLOP. Sie können die XMP-Metadaten auch entfernen, z.B. weil sie mit dem aktuellen Dokumentinhalt nicht mehr übereinstimmen. Dazu geben Sie in PLOP folgenden Aufruf ein:

```
plop --inputopt xmppolicy=remove --outfile output.pdf input.pdf
```

Beachten Sie, dass Standard-Identifikatoren (z.B. für PDF/A) verloren gehen, wenn die XMP-Metadaten entfernt werden.

Umgang mit ungültigen XMP-Metadaten. PDF-Dokumente enthalten manchmal ungültiges XMP, entweder auf XML- oder auf XMP/RDF-Ebene. PLOP weist solche Dokumente standardmäßig zurück und stoppt die Verarbeitung. Zur genaueren Analyse solcher Eingabedokumente können Sie die Option *xmppolicy* von *PLOP_open_document( )* verwenden. Damit lassen sich die folgenden Fälle unterscheiden:

- *xmppolicy=rejectinvalid:* Standardmäßig erzeugt PLOP bei ungültigem XMP keine PDF-Ausgabe.
- *xmppolicy=ignoreinvalid:* Ungültiges XMP wird ignoriert und der Text aus der Fehlermeldung der XML-Analyse wird in die erzeugte XMP-Ausgabe zur Unterstützung bei der Fehlersuche hinzugefügt. Beachten Sie, dass mit dieser Option keine PDF/A- oder PDF/X-3/4/5-Ausgabe erzeugt werden kann.
- *xmppolicy=remove:* entfernt das Eingabe-XMP. Damit lassen sich unerwünschte Metadaten entfernen.

Wenn Sie beispielsweise verhindern möchten, dass die Batch-Verarbeitung unterbrochen wird, können Sie Probleme durch ungültiges XMP im Eingabedokument folgendermaßen ignorieren:

```
plop --inputopt "xmppolicy=ignoreinvalid" --outfile output.pdf input.pdf
```

1.9 Details zur PDF-Verarbeitung mit PLOP

Zulässige Eingabedokumente. PLOP akzeptiert die folgenden PDF-Varianten:

- PDF 1.6 (Acrobat 7) und alle älteren Versionen
- PDF 1.7 (Acrobat 8), technisch identisch mit ISO 32000-1
- PDF 1.7 Adobe extension level 3 (Acrobat 9)
- PDF 1.7 Adobe extension level 8 (Acrobat X und höher)
- PDF 2.0 gemäß ISO 32000-2 (derzeit als Entwurf verfügbar)

Abhängig von der gewünschten Operation kann ein Kennwort für verschlüsselte Dokumente erforderlich sein. PLOP versucht, verschiedene Arten von beschädigten PDF-Dokumenten zu reparieren.

PDF-Version. Das generierte Ausgabedokument erhält eine PDF-Version, die mindestens so hoch wie die des Eingabedokuments ist und auch höher werden kann. PLOP verwendet die PDF-Version des Eingabedokuments, die sich gemäß folgender Regeln ändern kann:

- Im PDF/A-1- und PDF/X-Modus bleibt die PDF-Version unverändert; im PDF/A-2/3-Modus wird PDF 1.7 erzeugt.
- Andernfalls ist die PDF-Version der Ausgabe mindestens PDF 1.6.
- Kennwortschutz (Option *masterpassword)* erhöht die PDF-Version beim Verschlüsselungsalgorithmus 4 auf PDF 1.7ext3 und auf PDF 1.7ext8 beim Verschlüsselungsalgorithmus 11 (siehe »Verschlüsselungsalgorithmen und Schlüssellängen«, Seite 62).
- Zertifikatsicherheit (Funktion *PLOP_add_recipient( ))* erhöht die PDF-Version auf PDF 1.6 beim pCOS-Algorithmus 6 und auf PDF 1.7ext3 beim pCOS-Algorithmus 10 (siehe »PDF-Verschlüsselungsalgorithmus und Schlüssellänge«, Seite 82).
- Einige Signaturfunktionen erhöhen die PDF-Version auf PDF 1.7ext8 (siehe Tabelle 7.1).

Konformität zu den PDF-Standards. Bei der PLOP-Verarbeitung werden verschiedene PDF-Standards berücksichtigt. Entspricht das Eingabedokument einem der folgenden Standards, so ist dies auch für die von PLOP erzeugte Ausgabe gewährleistet:

- PDF/A-1/2/3: alle Varianten
- PDF/X-3/4/5 und PDF/VT-1/2: alle Varianten
- PDF/UA-1

Beachten Sie, dass einige PLOP-Operationen (vor allem Verschlüsselung) bei manchen Standards nicht zulässig sind. Verwenden Sie in diesen Fällen die Option *sacrifice*, um entsprechende Prioritäten festzulegen (siehe unten).

Aufgeben bestimmter Eigenschaften des Eingabe-PDFs. Manchmal ergeben sich Konflikte zwischen PDF-Dokumenteigenschaften und PLOP-Operationen. So dürfen PDF/A-Dokumente beispielsweise keine Verschlüsselung enthalten. Wie also soll PLOP vorgehen, wenn ein PDF/A-Dokument zu verschlüsseln ist? Im Standardfall weist PLOP die Operation zurück und löst eine Exception aus. Um die angeforderte Operation durchzuführen, obwohl es die Eingabe nicht zulässt, verwenden Sie die Option *sacrifice* von *PLOP_create_document( )*. Für das Kommandozeilen-Tool verwenden Sie die Option

--outputopt. Im obigen Beispiel wird der PDF/A-Konformitätseintrag aus dem Dokument entfernt, so dass eine Verschlüsselung möglich wird.

Es gibt verschiedene Operationen, die mit bestimmten Eigenschaften von Eingabedokumenten nicht vereinbar sind. In solchen Fällen können Sie generell die Option *sacrifice* verwenden, um eine Operation durch Aufgeben einer bestimmten Dokumenteigenschaft zu ermöglichen (für weitere Informationen siehe Tabelle 8.1):

- PDF/A: PLOP erstellt digitale Signaturen auf eine Weise, die zu PDF/A konform ist: Eingabedokumente, die dem Standard PDF/A-1, PDF/A-2 oder PDF/A-3 genügen, erzeugen auch nach dem Signieren PDF/A-konforme Ausgabe. Verschlüsselung ist für PDF/A-Dokumente jedoch nicht zulässig, da der Standard dies nicht erlaubt. Sie können jedoch die Option *sacrifice={pdfa}* nutzen, um die Konformität zu PDF/A aufzugeben. Zur Visualisierung von Signaturen verwendete PDF-Seiten müssen ebenfalls PDF/A-konform sein (siehe Abschnitt 7.3.1, »Visualisieren von Signaturen mit Grafik oder Logo«, Seite 105).
- PDF/X: PDF/X-1a/3/4/5 erlaubt weder Verschlüsselung noch auf der Seite sichtbare Signaturfelder. In diesen Fällen löst PLOP eine Exception aus, Sie können jedoch die Option *sacrifice={pdfx}* nutzen, um die Konformität zu PDF/X aufzugeben. Die Visualisierung von Signaturen wird im PDF/X-Modus nicht unterstützt.
- PDF/UA: die meisten PLOP-Operationen sind konform zu PDF/UA-1, außer *permissions=noaccessible*. Sie können die Option *sacrifice={pdfua}* nutzen, um die Konformität zu PDF/UA aufzugeben.
- PLOP kann ein Dokument nicht signieren und löst stattdessen eine Exception aus, wenn das Dokument Formularfelder ohne »Appearance Streams«, eine PDF-Datenstruktur für die visuelle Repräsentation von Feldern, enthält (z.B. Formularfelder, die mit PDFlib 7/8/9 erstellt wurden). Der Grund besteht darin, dass Acrobat die fehlenden Appearance-Streams für Formularfelder nachbilden müsste, was die Signatur sofort ungültig werden ließe. In diesem Fall können Sie die Option *sacrifice={fields}* von *PLOP_create_document( )* nutzen, um alle vorhandenen Formularfelder aufzugeben. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--outputopt*. Beachten Sie, das die Einschränkung für Formularfelder nicht für Signaturfelder gelten, die die erzeugte Signatur enthalten.
- Enthält ein unverschlüsseltes Dokument verschlüsselte Dateianhänge, für die kein Kennwort verfügbar ist, so wird die Verarbeitung standardmäßig abgebrochen. In diesem Fall können Sie die Option *sacrifice={fields}* von *PLOP_create_document( )* nutzen, um alle vorhandenen Formularfelder aufzugeben. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--outputopt*. Mit dieser Option werden alle verschlüsselten Dateianhänge entfernt, für die kein Kennwort verfügbar ist.
- Enthält das Eingabedokument eine oder mehrere digitale Signaturen und wird keine neue Signatur im Update-Modus erzeugt, wird die Verarbeitung standardmäßig mit einer Exception abgebrochen. In diesem Fall können Sie die Option *sacrifice={signatures}* von *PLOP_create_document( )* nutzen, um alle vorhandenen Signaturen aufzugeben. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--outputopt*.

Eigenschaften des Eingabedokuments, die generell verloren gehen. Folgende Eigenschaften des Eingabedokuments gehen bei der Verarbeitung durch PLOP verloren:

- Bei einem linearisierten Eingabedokument wird die Linearisierung entfernt. Mit der Option *linearize* von *PLOP_create_document( )* können Sie die Ausgabe linearisieren.

Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--linearize*. Beachten Sie, dass Linearisierung nicht mit digitalen Signaturen kombiniert werden kann.

- Dokumente mit erweiterter Reader-Funktionalität: die erweiterte Reader-Funktionalität dieser Dokumente geht bei der Verarbeitung mit PLOP verloren. Da Dokumente mit erweiterter Reader-Funktionalität nur mit Adobe-Software erstellt werden können, gibt es keine Lösung für dieses Problem.

Temporär erforderlicher Plattenspeicher. PLOP liest ein PDF-Eingabedokument und schreibt ein Ausgabe-PDF. Das Ausgabedokument ist etwa genauso groß wie das Eingabedokument (sofern die PLOP-Optimierung nicht redundante Informationen entfernt). Meist ist kein zusätzlicher Plattenplatz erforderlich. Bei Linearisierung oder digitalen Signaturen benötigt PLOP/PLOP DS jedoch zusätzlichen temporären Speicherplatz.

Temporärdateien werden standardmäßig im aktuellen Verzeichnis angelegt. Dies können Sie mit der Option *tempdirname* von *PLOP_create_document( )* ändern. Der für temporäre Daten erforderliche Plattenplatz entspricht im wesentlichen der Größe der Eingabedatei. Wird Linearisierung in Kombination mit direkter PDF-Generierung im Speicher (d.h. ohne Übergabe eines Namens für die Ausgabedatei) durchgeführt, benötigt PLOP dafür temporären Plattenplatz, der ungefähr doppelt so groß wie die Eingabe ist.

Große PDF-Dokumente. Auf den ersten Blick mag es nicht notwendig erscheinen, PDF-Dokumente im Bereich mehrerer Gigabytes anzulegen, doch gibt es Business-Anwendungen, die solche Dokumente erzeugen oder verarbeiten müssen, z.B. für eine große Anzahl von Rechnungen oder Kontoauszügen. Während bei PLOP die Größe der erzeugten Dokumente nicht nach oben begrenzt ist, werden von der PDF-Referenz und einigen PDF-Standards mehrere Einschränkungen vorgegeben:

- Begrenzung der Dateigröße auf 2 GB: Bei PDF/A und anderen Standards ist die Dateigröße auf 2 GB begrenzt. Wenn ein Dokument größer als 2 GB ist, löst PLOP bei der Erzeugung von PDF/A-, PDF/X-4- und PDF/X-5-Ausgabe eine Exception aus. Für andere Ausgabe können Dokumente größer als 2 GB erzeugt werden.
- Begrenzung der Dateigröße auf 10 GB: PDF-Dokumente wurden lange Zeit intern von Querverweis-Tabellen auf 10 Dezimalstellen und damit 10^{10}-1 Bytes begrenzt, was in etwa 9,3 GB entspricht. Allerdings kann diese Grenze mit komprimierten Objekt-Streams überschritten werden. Während komprimierte Objekt-Streams die Dateigröße ohnehin verringern, unterliegen die komprimierten Querverweis-Streams, die Teil der Implementierung von *objectstreams* sind, nicht länger der Grenze von 10 Dezimalstellen und ermöglichen damit die Erstellung von PDF-Dokumenten über 10 GB hinaus.
- Anzahl der Objekte: Während die Anzahl der Objekte in einem Dokument durch PDF generell nicht begrenzt wird, limitieren die Standards PDF/A, PDF/X-4 und PDF/X-5 die Zahl der indirekten Objekte in einem Dokument auf 8.388.607. Wenn ein Dokument mehr Objekte benötigt, löst PLOP bei der Erzeugung von PDF/A-, PDF/X-4- und PDF/X-5-Ausgabe eine Exception aus. In anderen Modi können Dokumente mit einer höheren Objektzahl immer erzeugt werden. Diese Überprüfung kann mit der Option *limitcheck=false* deaktiviert werden.

In PLOP nicht unterstützte Funktionalität. Beachten Sie folgende Einschränkungen:

- PLOP ist kein Cracker-Tool – es kann nicht verwendet werden, um sich Zugriff auf geschützte Dokumente ohne entsprechendes Master-Kennwort zu verschaffen.

- Sie können keine dynamischen XFA-Formulare verarbeiten, da es sich nicht um echte PDF-Dokumente handelt, sondern um in eine dünne PDF-Schicht verpackte XML-Formulare.

2 Funktionsumfang von PLOP DS (Digital Signature)

Hinweis Digitale Signaturen werden nur von PLOP DS, nicht aber vom Basisprodukt PLOP unterstützt.

Eine detaillierte Beschreibung digitaler Signaturen für PDF-Dokumente finden Sie in Kapitel 7, »Digitale Signaturen mit PLOP DS«, Seite 89. Der vorliegende Abschnitt gibt lediglich einen kurzen Überblick sowie einige einführende Beispiele.

2.1 Signaturfunktionen in PLOP DS

Steuerung der PDF-Signatur.

- Erstellung von Signaturen in bestehenden PDF-Signaturfeldern oder Generierung neuer Felder für die Signatur. Die Signaturen können an einer bestimmten Stelle auf der Seite sichtbar oder unsichtbar sein.
- Visualisierung der digitalen Signatur durch Import eines Logos, den Scan einer manuellen Unterschrift oder eine andere Darstellung als PDF-Seite.
- Erstellung zertifizierter PDF-Dokumente (auch Zertifizierungs- oder Autorensignatur genannt), die Dokumentänderungen ermöglichen (zum Beispiel Formularfelder ausfüllen), ohne die Signatur ungültig zu machen.
- Validierungsinformationen können direkt in der Signatur gemäß ISO 32000-1 gespeichert werden oder in einem Document Security Store (DSS) gemäß ISO 32000-2 und PAdES Teil 4.
- Signaturen können in einem inkrementellen PDF-Update angewendet werden, um vorhandene Signaturen und Dokumentstruktur zu bewahren oder durch Überschreiben der Dokumentstruktur zur Optimierung und Verschlüsselung.

PDF-Versionen und PDF-Standards. PLOP DS unterstützt alle relevanten PDF-Versionen und -Standards:

- PLOP DS verarbeitet alle PDF-Versionen bis hin zu Acrobat DC, also PDF 1.7 (ISO 32000-1) einschließlich Extension Level 8. Auch für den Standard PDF 2.0 (ISO 32000-2) kann PLOP DS bereits Dokumente verarbeiten.
- PLOP DS berücksichtigt die Archivierungsstandards PDF/A-1/2/3 (ISO 19005): ist das Eingabedokument PDF/A-konform, so ist dies auch für die Ausgabe gewährleistet. Auch XMP Extension Schemas gemäß PDF/A werden von PLOP DS vollständig unterstützt. Die Fähigkeit, PDF/A-konforme Metadaten in PDF-Dokumente einzufügen, ist ein wichtiger Vorteil von PLOP DS.
- Entsprechend unterstützt PLOP DS auch die Standards zur Druckverarbeitung PDF/X-1a/3/4/5 (ISO 15930), den Standard für Transaktionsdruck PDF/VT-1/2 (ISO 16612-2) sowie PDF/UA-1 (ISO 14289) für barrierefreie PDF-Dokumente.

Signaturstandards.

- CMS-basierte PDF-Signaturen gemäß ISO 32000-1
- Signaturen für Langzeitvalidierung (Long-Term Validation, LTV) gemäß ISO 32000-2
- PAdES *(PDF Advanced Electronic Signatures)* gemäß ETSI TS 102 778 Teil 2, 3 und 4, ETSI EN 319 142 und CAdES (ETSI TS 101 733)

PAdES-Signaturstufen.

- Einfache Signatur (PAdES-Signaturstufe B-B)
- Signatur mit Zeit (PAdES-Signaturstufe B-T)
- Signatur mit Material für die Langzeitvalidierung (PAdES-Signaturstufe B-LT)
- Signatur mit Langzeitverfügbarkeit und Integrität von Validierungsmaterial (PAdES-Signaturstufe B-LTA)
- Einfache elektronische Signatur (PAdES Stufe E-BES) und explizit Richtlinien-basierte Elektronische Signatur (PAdES Stufe E-EPES) gemäß PAdES Teil 3

Kryptografische Details der Signatur.

- Signaturen gemäß den RSA- und DSA-Verfahren sowie Elliptic Curve Digital Signature Algorithm (ECDSA) basierend auf Kryptografie mit elliptischen Kurven. Die von NIST und Brainpool empfohlenen elliptischen Kurven werden unterstützt.
- Starke Signaturen und Hashfunktionen.
- Einbindung der vollständigen Zertifikatskette in die erzeugten Signaturen: Das bedeutet, dass Signaturen mit Zertifikaten einer CA *(Certificate Authority)* auf der Adobe Approved Trust List (AATL) oder der European Union Trust List (EUTL) auf Client-Seite ohne weitere Konfiguration in Acrobat und Adobe Reader validiert werden können.
- Einbindung von OCSP-Antworten *(Online Certificate Status Protocol* gemäß RFC 2560 und RFC 6960) und Zertifikatsperrlisten *(Certificate Revocation Lists,* CRL gemäß RFC 3280) als Sperrinformationen für die Langzeitvalidierung *(Long-Term Validation,* LTV).

Zeitstempel.

- Anfordern eines Zeitstempels von einer vertrauenswürdigen Zeitquelle (*Time-Stamp Authority, TSA)* gemäß RFC 3161, RFC 5816 und ETSI EN 319 422 sowie Einbetten in die erzeugte Signatur. Die Daten der TSA können aus AATL-Zertifikaten entnommen werden, um Zeitstempel ohne weitere Konfiguration zu erstellen.
- Erstellen von Zeitstempeln auf Dokumentebene gemäß ISO 32000-2 und PAdES Teil 4. Ein Zeitstempel auf Dokumentebene garantiert den Zustand des Dokuments auch ohne persönliche Signatur.
- Unterstützung für den Parameter *policy* des Zeitstempel-Protokolls sowie alle gebräuchlichen Hashfunktionen für Zeitstempel.

Signatur-Engines. PLOP DS unterstützt verschiedene kryptografische Engines, also Komponenten zur Generierung digitaler Signaturen:

- Die integrierte Engine implementiert die erforderlichen kryptografischen Funktionen direkt in PLOP DS ohne externe Abhängigkeiten. Die integrierte Engine unterstützt Software-basierte digitale IDs in den verbreiteten Zertifikatformaten PKCS#12 und PFX.
- PLOP DS kann kryptografische Tokens über die PKCS#11-Schnittstelle verbinden. Zur Signaturerstellung können daher auch digitale IDs auf Smartcards, USB-Sticks und anderen sicheren Geräten genutzt werden, einschließlich Geräten mit integrierter Tastatur zur sicheren PIN-Eingabe.
- Die PKCS#11-Schnittstelle kann auch zum Signieren mit einem Hardware-Security-Modul (HSM) verwendet werden. HSMs bieten sichere Schlüsselspeicher und genügend Leistung für hochvolumige Signaturanwendungen. PLOP DS verwendet PKCS#11-Sessions, um die Leistung bei Massensignaturen mit HSMs zu maximieren.

- Unter Windows kann PLOP DS das Microsoft Cryptographic API (CAPI) als kryptografische Engine verwenden und damit die kryptografische Infrastruktur von Windows nutzen. Digitale IDs aus dem Zertifikatspeicher von Windows können zur Signaturerstellung genutzt werden. Dabei lassen sich sowohl Software-basierte digitale IDs als auch sichere Hardware-Tokens einsetzen. Beachten Sie, dass für die MSCAPI-Engine nicht alle Signaturfunktionen, wie zum Beispiel LTV, verfügbar sind.

In PLOP DS nicht unterstützte Funktionalität. Beachten Sie folgende Einschränkungen:

- Sie können mit PLOP DS keine PDF-Dokumente mit erweiterter Reader-Funktionalität erstellen (z.B. Anmerkungen erstellen in Adobe Reader erlauben), da dazu eine bestimmte Adobe-Signatur benötigt wird.
- Sie können weder statische noch dynamische XFA-Formulare signieren.

2.2 Evaluierung von PLOP und PLOP DS

Installation des Zertifikats der PDFlib Demo CA in Acrobat. Der folgende Schritt ist zur Erstellung digitaler Signaturen mit PLOP DS nicht erforderlich. Wenn Sie PLOP DS allerdings mit den Beispiel-Zertifikaten aus dem Produktpaket evaluieren, sollten Sie Acrobat wie unten beschrieben konfigurieren. Wenn Sie mit Zertifikaten einer kommerziellen CA arbeiten, die in der Liste der vertrauenswürdigen Zertifikate von Acrobat installiert ist, ist dies nicht erforderlich (siehe Abschnitt 7.1.3, »Vertrauenswürdige Stammzertifikate in Acrobat (Trusted Root Certificates)«, Seite 93).

Die Beispielzertifikate im PLOP DS-Paket wurden von der PDFlib Demo CA erzeugt und signiert. Wenn Sie das selbstsignierte Stammzertifikat dieser CA für Acrobat verfügbar machen, werden die erzeugten Signaturen in Acrobat als voll gültig akzeptiert. Gehen Sie zur Installation des Zertifikats der PDFlib Demo CA in Acrobat XI/DC folgendermaßen vor:

- Wählen Sie *Bearbeiten, Voreinstellungen..., Unterschriften, Identitäten & vertrauenswürdige Zertifikate, Weitere..., Vertrauenswürdige Zertifikate, Importieren, Auswählen...*
- Gehen Sie zu *bind/data/PDFlibDemoCA_G2.crt* (Teil der PLOP-Installation) und klicken Sie auf *Importieren, OK*.
- Der Eintrag *PDFlib GmbH Demo CA G2* erscheint nun in de Liste der vertrauenswürdigen Zertifikate. Wählen Sie diesen Eintrag aus und klicken Sie auf *Bearbeiten*. Im Dialogfenster *Zertifikatberechtigung bearbeiten* aktivieren Sie *Dieses Zertifikat als vertrauenswürdigen Stamm verwenden* und *Zertifizierte Dokumente* und klicken auf *OK*.

Importieren digitaler Demo-IDs in Windows. Um die MSCAPI-basierte Signature-Engine von PLOP DS unter Windows zu testen, müssen Sie dem Zertifikatspeicher von Windows digitale IDs zur Verfügung stellen. Klicken Sie zum Import der digitalen IDs auf die entsprechende *.p12*-Datei. Der Zertifikatimport-Assistent öffnet sich und führt Sie durch die erforderlichen Schritte.

2.3 Signieren von Dokumenten mit PLOP DS

Zur Erstellung einer digitalen Signatur wird eine digitale ID benötigt, die zum Beispiel als Datei vorliegt, aus dem Zertifikatspeicher von Windows stammt oder von einem kryptografischen Token (z.B. einer Smartcard oder einem USB-Stick). Während für den

Zugriff auf die digitale ID in einer Datei ein Kennwort erforderlich ist, wird der Zertifikatspeicher von Windows in der Regel durch die Windows-Anmeldung abgesichert, so dass hier kein Kennwort notwendig ist. Kryptografische Tokens sind meist mit einer PIN geschützt, die beim Signieren entweder durch die Software oder direkt über die integrierte Tastatur des Tokens übergeben wird.

Mit *PLOP_prepare_signature( )* können Sie eine digitale Signatur durch Angabe verschiedener Optionen vorbereiten und sie dann mit *PLOP_create_document( )* erstellen. Codebeispiele zum Signieren von PDF-Dokumenten finden Sie in den Minibeispielen *sign* und *multisign*, die in allen PLOP-Paketen enthalten sind. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--signopt*.

Grundlegende Beispiele für Signatur-Optionslisten. Eine unsichtbare Signatur für ein PDF-Dokument mit Hilfe einer digitalen ID aus der Datei *demo_signer_rsa_2048.p12* erstellen. Das Kennwort *demo* für die digitale ID befindet sich in der Datei *pw.txt*:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt" ↵
       --outfile signed.pdf input.pdf
```

(Nur für Windows) Erzeugen einer unsichtbaren Signatur für ein PDF-Dokument mit Hilfe eines Zertifikats aus dem Zertifikatspeicher von Windows (aus dem Standardspeicher *My)*. Das Beispiel geht davon aus, dass die digitale ID durch die Windows-Anmeldung geschützt ist, so dass kein Kennwort benötigt wird:

```
plop --signopt "engine=mscapi digitalid={store=My subject={PLOP Demo Signer RSA-2048}}" ↵
        --outfile signed.pdf input.pdf
```

(Nur für Plattformen mit PKCS#11-Unterstützung) Erzeugen einer unsichtbaren Signatur für ein PDF-Dokument mit Hilfe einer digitalen ID von einem kryptografischen Token. Die PKCS#11-Schnittstelle für den Token ist in der Bibliothek *cryptoki.dll* implementiert, die vom Smartcard-Hersteller bereitgestellt werden muss. Das Kennwort für die digitale ID befindet sich in der Datei *pw.txt*:

```
plop --signopt "engine=pkcs#11 digitalid={filename=cryptoki.dll} passwordfile=pw.txt" ↵
       --outfile signed.pdf input.pdf
```

Für weitere Informationen siehe Abschnitt 7.2, »Signieren von Dokumenten mit PLOP DS«, Seite 96.

2.4 Zertifizierungssignaturen

Eine Zertifizierungs- oder Autorensignatur bescheinigt den Status des Dokuments, wie der Unterzeichner es erstellt hat, und ermöglicht gleichzeitig bestimmte Dokumentänderungen ohne die Zertifizierungssignatur ungültig zu machen. Mit der Option *certification* lassen sich die zulässigen Änderungen an einem zertifizierten Dokument festlegen, z.B. Ausfüllen von Formularfeldern erlaubt:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
           certification=formfilling" ↵
           --outfile certified.pdf input.pdf
```

2.5 Zeitstempel

Um einer Signatur einen Zeitstempel hinzuzufügen, benötigen Sie die URL eines Zeitstempeldienstes *(Time-Stamp Authority, TSA),* die Sie an die Option *timestamp* übergeben müssen:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
        timestamp={source={url={http://timestamp.acme.com/tsa-noauth/tsa}}}" ↵
        --outfile signed.pdf input.pdf
```

Auf ähnliche Weise kann ein Zeitstempel auf Dokumentebene mit der Option *doctimestamp* übergeben werden:

```
plop --signopt ↵
        "doctimestamp={source={url={http://timestamp.acme.com/tsa-noauth/tsa}}}" ↵
        --outfile signed.pdf input.pdf
```

Für weitere Informationen siehe Abschnitt 7.5, »Zeitstempel«, Seite 123.

2.6 Signaturen für Langzeitvalidierung (LTV)

Für die Langzeitvalidierung *(Long-Term Validation, LTV)* müssen alle Zertifikate in der Zertifikatskette verfügbar sein und die Zertifikatsperrlisten online oder von einer Datei auf der Festplatte abrufbar sein, wenn die Signatur erstellt wird. Hierzu müssen von der *Public Key Infrastructure* (PKI) geeignete OCSP- oder CRL-Server zur Verfügung gestellt werden. In vielen Fällen, vor allem bei AATL-Zertifikaten, kann die erforderliche Netzwerkinformation aus dem Signaturzertifikat gelesen werden. Andernfalls müssen Sie die entsprechende Netzwerk-Ressource mit den Optionen *ocsp* und/oder *crl/crlfile/crldir* übergeben. Um die gesamte Zertifikatskette zur Verfügung zu stellen, müssen Sie die Option *rootcertfile* mit dem Namen einer PEM-Datei angeben, die das Stammzertifikat der CA enthält.

Bei Signaturen für die Langzeitvalidierung benötigt man meist PKI-Ressourcen (CRL oder OCSP) für das verwendete Zertifikat, die in den Demo-Zertifikaten von PLOP DS nicht enthalten sind. Stattdessen können Sie die CRL-Datei *PDFlibDemoCA_G2.crl* aus der PLOP-Distribution verwenden (diese Zertifikatsperrliste hat eine sehr lange Gültigkeit, was im produktiven Einsatz unzulässig wäre). Für das PLOP-Kommandozeilen-Tool verwenden Sie folgenden Aufruf zur Erstellung von Signaturen für die Langzeitvalidierung:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} password=demo ltv=full ↵
        crlfile=PDFlibDemoCA_G2.crl rootcertfile=PDFlibDemoCA_G2.pem" ↵
        --outfile ltv-signed.pdf input.pdf
```

Beim nächsten Beispiel gehen wir davon aus, dass die erforderliche OCSP- oder CRL-Adresse wie bei den meisten kommerziellen Zertifikaten im Signaturzertifikat vorhanden ist. Übergeben Sie die Option *ltv=full*, um sicherzustellen, dass eine Signatur für die Langzeitvalidierung erstellt werden kann:

```
plop --signopt "digitalid={filename=signer.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=RootCA.pem" --outfile ltv-signed.pdf input.pdf
```

Abhängig von der beteiligten PKI können noch weitere Angaben erforderlich sein, insbesondere müssen Sperrinformationen zu den Zertifikaten auch für den OCSP/CRL-Unterzeichner und die TSA verfügbar sein.

2.7 PAdES-Signaturen

Die PAdES-Signaturstandards verbessern PDF-Signaturen und gewährleisten das Einhalten von EU-Richtlinien. Mit verschiedenen Optionen lassen sich Signaturen gemäß unterschiedlicher PAdES-Varianten erstellen. Mit der folgenden Kommandozeile können Sie beispielsweise eine einfache Signatur gemäß PAdES Teil 3 (PAdES-Signaturstufe B-B) erstellen:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
        --outfile signed.pdf input.pdf
```

Mit der folgenden Kommandozeile können Sie eine Signatur gemäß PAdES Teil 3 mit explizitem Richtlinien-Identifikator erstellen (PAdES-Signaturstufe E-EPES):

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
        policy={oid=2.16.276.1.89.1.1.1.1.3 commitmenttype=origin}" ↵
        --outfile signed.pdf input.pdf
```

Für weitere Informationen siehe Abschnitt 7.7, »Die Signaturstandards CAdES und PAdES«, Seite 134.

2.8 Visualisierung digitaler Signaturen

Eine digitale Signatur kann zum Beispiel durch ein Firmenlogo oder durch eine handschriftliche Unterschrift visualisiert werden. Die Signaturvisualisierung müssen Sie als PDF-Dokument übergeben, welches in das Signaturfeld eingefügt wird. Falls das Dokument noch kein Signaturfeld enthält, müssen Sie geeignete Formularfeld-Koordinaten übergeben. Mit der folgenden Kommandozeile platzieren Sie das Signaturbild *signing_man.pdf* in das Feldrechteck:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
        field={name=Signature1 rect={10 10 adapt adapt}}" --visdoc signing_man.pdf ↵
        --outfile signed.pdf input.pdf
```

Für weitere Informationen siehe Abschnitt 7.3.1, »Visualisieren von Signaturen mit Grafik oder Logo«, Seite 105.

2.9 Abfrage von Signatureigenschaften

Mit der Programmierschnittstelle pCOS, die in PLOP DS integriert ist, lassen sich die Signatureigenschaften eines PDF-Dokuments abfragen. Das pCOS Cookbook-Topic *interactive_elements/signatures* zeigt, wie sich Signaturtypen und -details abfragen lassen. Ein Codebeispiel zur Abfrage von Dokumentinformationen mit pCOS finden Sie im Minibeispiel *dumper,* das in allen PLOP-Paketen enthalten ist. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--info* (siehe Abschnitt 1.6, »Abfrage von Dokumentinformationen«, Seite 23):

```
plop --info *.pdf
```

Der Programmaufruf bewirkt in etwa folgende Ausgabe:

```
       File name: hellosign.pdf
     PDF version: 1.7
      Encryption: No encryption
      ...
      Tagged PDF: false
      Signatures: 1
        signature field 'Signature1': invisible approval signature, CAdES
Reader-enabled: false
```

3 PLOP- und PLOP DS-Kommandozeilen-Tool

3.1 PLOP- und PLOP DS-Kommandozeilen-Optionen

Das kombinierte Kommandozeilen-Tool für PLOP und PLOP DS ermöglicht es Ihnen, PDF-Dokumente ohne Programmieraufwand zu verschlüsseln, entschlüsseln, optimieren, reparieren oder zu signieren. Außerdem können Sie das Kommandozeilen-Tool nutzen, um den Status von PDF-Dokumenten abzufragen. Das Programm PLOP kann mit einer Reihe von Kommandozeilen-Optionen gesteuert werden. Es wird für eine oder mehrere PDF-Eingabedateien folgendermaßen aufgerufen (Elemente in eckigen Klammern sind optional):

```
plop --help
plop [ <general options> ] --info [ --outfile <filename> ] <filename> ...
plop [ <general options> ] <transform options> --outfile <filename> <filename>
plop [ <general options> ] <transform options> --targetdir <pathname> <filename>...
```

Das PLOP-Kommandozeilen-Tool baut auf der PLOP-Bibliothek auf. PLOP repariert alle Eingabedokumente, die als beschädigt erkannt werden. Mit den Optionen *--inputopt*, *--outputopt*, *--plopopt*, *--signopt* und *--visdocopt* können Sie Optionen an die Bibliothek übergeben. Die entsprechenden Optionslisten finden Sie in Kapitel 8, »API-Referenz für die PLOP- und PLOP DS-Bibliothek«, Seite 139. Tabelle 3.1 zeigt eine Liste aller PLOP-Kommandozeilen-Optionen.

Tabelle 3.1 PLOP-Kommandozeilen-Optionen

Option	Parameter	Funktion
--		*Beendet die Optionsliste; nützlich, wenn Dateinamen mit dem Zeichen »-« beginnen.*
@filename[1]		*Festlegen einer Response-Datei mit dem Namen* `filename`*, die Optionen enthält; für eine Syntaxbeschreibung siehe »Response-Dateien«, Seite 42. Response-Dateien werden nur vor der Option `--` und dem ersten Dateinamen erkannt, können aber nicht verwendet werden, um den Parameter für eine andere Option zu ersetzen.*
--help, -? ***(oder keine Option)***		*Anzeige der Hilfe mit einer Übersicht über die verfügbaren Optionen.*
--info, -i		*Zeigt Statusinformationen der Eingabedatei an; es wird keine PDF-Ausgabe erstellt.*
--inputopt	*<Optionsliste>*	*Optionsliste für PLOP_open_document( ) (siehe Tabelle 8.3, Seite 146)*
--master, -m	*<Kennwort>*	*Master-Kennwort für die Ausgabedatei; eine fehlende Option wird als kein Kennwort interpretiert*
--noreplace, -n		*Existiert die Ausgabedatei bereits, wird sie nicht überschrieben, sondern es wird eine Exception ausgelöst. Standardwert: vorhandene Ausgabedateien werden überschrieben.*

Tabelle 3.1 PLOP-Kommandozeilen-Optionen

Option	Parameter	Funktion
--outfile, -o	<*Dateiname*>	*(Es muss genau ein Eingabedokument übergeben werden, sofern nicht auch* `--info` *angegeben ist; entweder* `--outfile` *oder* `--targetdir` *muss übergeben werden) Name der Ausgabedatei; dieser darf nicht mit dem Namen der Eingabedatei übereinstimmen.*
--outputopt	<*Optionsliste*>	*Optionsliste für PLOP_create_document( ) (siehe Tabelle 8.5, Seite 150)*
--password, -p	<*Kennwort*>	*Benutzer- oder Master-Kennwort für verschlüsselte Dokumente. Das übergebene Kennwort wird für alle Eingabedokumente verwendet. Eingabedokumente, die unterschiedliche Kennwörter benötigen, müssen in getrennten Programmaufrufen verarbeitet werden.* *Wurde eine digitale ID mit* `--inputopt` *übergeben, wird das Kennwort auf die ID angewendet.*
--permissions	<*Berechtigungen*>	*(Benötigt* `--master` *oder* `--recipient`*) Liste der Zugriffsberechtigungen für das Ausgabedokument. Sie kann eine beliebige Anzahl der Schlüsselwörter* `noprint`, `nomodify`, `nocopy`, `noannots`, `noassemble`, `noforms`, `noaccessible`, `nohiresprint` *und* `plainmetadata` *enthalten (siehe Tabelle 5.3, Seite 66). Daneben kann folgendes Schlüsselwort verwendet werden (Standardwert: keine Zugriffsbeschränkungen):* ***keep*** *Behält die Berechtigungseinstellungen des Eingabedokuments bei. Dieses Schlüsselwort kann um obige Schlüsselwörter ergänzt werden, um die Berechtigungseinstellungen des Eingabe-PDFs zu modifizieren, z.B.* `keep noprint`. *Im Zertifikatsicherheit-Modus (Option* `-recipient`*) ist nur* `plainmetadata` *erlaubt. Weitere Berechtigungsbeschränkungen können in der Option* `permissions` *der Optionsliste* `--recipient` *festgelegt werden.*
--plopopt	<*Optionsliste*>	*Optionsliste für PLOP_set_option( ) (siehe Tabelle 8.13, Seite 169). Damit können die Optionen* `license` *oder* `licensefile` *übergeben werden.*
--recipient, -r[1]	<*Optionsliste*>	*Optionsliste für PLOP_add_recipient( ) zum Hinzufügen eines Empfängers für ein Dokument mit Zertifikatsicherheit. Wird diese Option mindestens einmal übergeben, wird der Modus Zertifikatsicherheit aktiviert. Dieser Empfänger wird für alle Eingabedateien verwendet. Diese Option darf nicht mit* `--master/-m` *und* `--user/-u` *kombiniert werden.*
--recsize	<*Blockgröße*>	*(Nur für MVS) Record-Größe der Ausgabedatei. Standardwert: 0 (Ausgabe ohne feste Blockgröße)*
--searchpath, -s[1]	<*Pfad*>	*Name eines Verzeichnisses, in dem Dateien gesucht werden. Der Pfadname darf nicht mit einem Minuszeichen »-« beginnen (stellen Sie bei Bedarf* **.** */ voran). Standardwert: aktuelles Verzeichnis*
--signopt, -S	<*Optionsliste*>	*(Nur für PLOP DS) Optionsliste für PLOP_prepare_signature( ) zur digitalen Signierung von Dokumenten (siehe Tabelle 8.8, Seite 157).*
--targetdir, -t	<*Verzeichnisname*>	*(Einer der Werte* `--outfile` *oder* `--targetdir` *muss übergeben werden) Name des Ausgabeverzeichnisses; das Verzeichnis muss bereits vorhanden sein.*
--tempdirname	<*Verzeichnisname*>	*Name eines Verzeichnisses für die temporären Dateien, die für die PLOP-interne Verarbeitung benötigt werden. Ist diese Option leer, werden temporäre Dateien im aktuellen Verzeichnis abgelegt. Standardwert: leer*
--tempfilename, -T	<*Dateiname*>	*(nur für MVS) Vollständiger Name einer temporären Datei, die für die PLOP-interne Verarbeitung erforderlich ist. Ist diese Option leer, generiert PLOP selbst einen eindeutigen Namen. Der Benutzer muss die temporäre Datei nach Ausführung von PLOP löschen. Standardwert: leer*

Tabelle 3.1 PLOP-Kommandozeilen-Optionen

Option	*Parameter*	*Funktion*
--user, -u	*<Kennwort>*	*(Erfordert* `--master`*) Benutzerkennwort für die Ausgabe; keine Option bedeutet kein Kennwort*
--verbose, -v	0, 1, 2, 3	*Verbosity-Level (Standardwert: 1):* ***0*** *Keine Informationsausgabe* ***1*** *Nur Fehlermeldungen werden ausgegeben* ***2*** *Dateinamen und API-Funktionen werden den Fehlermeldungen hinzugefügt* ***3*** *detaillierter Report*
--visdoc	*<Dateiname>*	*(Nur mit* `--signopt`*) Name der PDF-Datei, aus der eine Seite zur Visualisierung der digitalen Signatur verwendet wird.*
--visdocopt	*<Optionsliste>*	*(Nur mit* `--visdoc`*) Optionsliste für PLOP_open_document( ) (siehe Tabelle 8.3, Seite 146) zum Öffnen des Dokuments mit dem Signaturbild*
--webopt, -w		*Linearisierung der PDF-Ausgabe zur Verteilung im Web, auch Web-Optimierung genannt. Standardwert: keine Linearisierung*

1. Diese Option kann mehrfach übergeben werden.

Erstellen von PLOP-Kommandozeilen. Beim Erstellen von PLOP-Kommandozeilen sind folgende Regeln zu beachten:

- Eingabedateien werden in allen Verzeichnissen gesucht, die im *searchpath* festgelegt wurden.
- Für manche Optionen stehen Kurzformen zur Verfügung, die mit langen Optionen kombiniert werden können.
- Lange Optionen können abgekürzt werden, sofern die Abkürzung eindeutig ist (z.B. *--plop* statt *--plopopt)*.
- Wird eine Option mehrfach übergeben, so wird nur der letzte Wert berücksichtigt. Dies gilt jedoch nicht für Optionen, die in Tabelle 3.1 als »Kann mehrmals übergeben werden« gekennzeichnet sind.
- Abhängig vom Verschlüsselungsstatus der Eingabedatei wird ein Benutzer- oder Master-Kennwort für die Verarbeitung benötigt. Dieses wird mit der Option *--password* übergeben. PLOP überprüft, ob das Kennwort für die gewünschte Aktion gemäß Tabelle 5.2 ausreicht, und gibt gegebenenfalls eine Fehlermeldung aus.

Vor der Verarbeitung einer Datei prüft PLOP die gesamte Kommandozeile. Liegt in der Kommandozeile ein Syntaxfehler in einer übergebenen Option vor, werden keinerlei Dateien verarbeitet. Kann eine Datei nicht verarbeitet werden (z.B. weil das erforderliche Kennwort fehlt), so gibt PLOP eine Fehlermeldung aus und fährt mit der Verarbeitung der übrigen Dateien fort.

Dateinamen. Dateinamen mit Leerzeichen müssen beim Einsatz mit Kommandozeilen-Tools wie PLOP besonders behandelt werden. Um einen Dateinamen mit Leerzeichen zu verarbeiten, sollten Sie den gesamten Dateinamen in doppelte Anführungszeichen " setzen. Wildcards können auf die übliche Art verwendet werden. Zum Beispiel umfasst **.pdf* alle Dateinamen mit dem Suffix *.pdf* im Dateinamen. Beachten Sie, dass manche Systeme nach Groß- und Kleinschreibung unterscheiden (d.h. **.pdf* kann von **.PDF* verschieden sein). Beachten Sie außerdem, dass unter Windows keine Wildcards

für Dateinamen mit Leerzeichen verwendet werden können. Wildcards werden nur im aktuellen Verzeichnis und nicht für ein beliebiges *searchpath*-Verzeichnis ausgewertet.

Unter Windows akzeptieren alle Dateinamen-Optionen Unicode-Strings, z.B. wenn Dateien aus dem Windows Explorer in ein Kommandozeilen-Fenster gezogen werden.

Response-Dateien. Optionen können direkt in der Kommandozeile oder in einer Response-Datei übergeben werden. Die Inhalte einer Response-Datei werden an der Stelle in der Kommandozeile eingefügt, an der die Option *@filename* gefunden wurde.

Eine Response-Datei ist eine einfache Textdatei mit Optionen und Parametern. Es müssen folgende Syntaxregeln eingehalten werden:

- Optionswerte müssen durch Leerraum, also Leerzeichen, Zeilenumbruch oder Tabulatorzeichen voneinander getrennt werden.
- Werte mit Leerzeichen müssen in doppelte Anführungszeichen eingeschlossen werden: "
- Doppelte Anführungszeichen am Anfang und Ende eines Wertes werden übersprungen.
- Ein doppeltes Anführungszeichen muss mit einem Backslash maskiert werden, wenn es als Anführungszeichen verwendet werden soll: \"
- Ein Backslash muss mit einem weiteren Backslash maskiert werden, wenn er als solcher verwendet werden soll: \\

Response-Dateien können verschachtelt werden, das heißt, *@filename* kann in einer anderen Response-Datei verwendet werden.

Response-Dateien können Unicode-Strings für Dateinamen- und Kennwort-Optionen enthalten. Response-Dateien können im Format UTF-8, EBCDIC-UTF-8 oder UTF-16 kodiert sein und müssen mit dem zugehörigen BOM beginnen. Wird kein BOM gefunden, werden die Inhalte der Response-Datei unter zSeries als EBCDIC interpretiert und auf allen anderen Plattformen als ISO 8859-1 (Latin-1).

Rückgabewerte. Das PLOP-Kommandozeilen-Tool gibt einen Wert zurück, mit dem geprüft werden kann, ob die angeforderten Operationen erfolgreich ausgeführt werden konnten:

- Rückgabewert 0: alle Kommandozeilen-Optionen und Eingabedateien konnten erfolgreich und vollständig verarbeitet werden.
- Rückgabewert 1: Bei der Dateiverarbeitung sind Fehler aufgetreten, die Verarbeitung wurde aber fortgeführt.
- Rückgabewert 2: In den Kommandozeilen-Optionen wurde ein Fehler gefunden. Die Verarbeitung wurde bei der fehlerhaften Option abgebrochen; es wurden keine Dokumente verarbeitet.

3.2 Beispiele für PLOP- und PLOP DS-Kommandozeilen

Die folgenden Beispiele zeigen einige nützliche Kombinationen von PLOP-Kommandozeilen-Optionen. Die Beispiele werden in zwei Varianten dargestellt; in der ersten wird die Langform für alle Optionen verwendet, in der zweiten die entsprechende Option im Kurzformat. In den folgenden Abschnitten finden Sie weitere Beispiele:

- Kapitel 1, »Funktionsumfang von PLOP«, Seite 17 (verschiedene Abschnitte);
- Abschnitt 5.3, »Anwenden von Kennwortschutz auf der Kommandozeile«, Seite 68;
- Abschnitt 6.5, »Anwenden von Zertifikatsicherheit auf der Kommandozeile«, Seite 86;
- Abschnitt 7.2, »Signieren von Dokumenten mit PLOP DS«, Seite 96.

Sicherheitsrelevante und andere Informationen zu allen PDF-Dateien im aktuellen Verzeichnis anzeigen:

```
plop --info *.pdf
plop -i *.pdf
```

Alle PDF-Dokumente eines Verzeichnisses linearisieren (unter der Annahme, dass diese kein Kennwort benötigen) und die Ergebnisdokumente in das Verzeichnis *output* kopieren. Mit dem Verbosity-Level 2 werden die Namen aller Ein- und Ausgabedateien bei der Verarbeitung ausgegeben:

```
plop --verbose 2 --webopt --targetdir output *.pdf
plop -v 2 -w -t output *.pdf
```

Alle Dateien im aktuellen Verzeichnis mit dem selben Benutzerkennwort *demo* und dem Master-Kennwort *DEMO* verschlüsseln und die Ausgabedateien ins Verzeichnis *output* kopieren:

```
plop --targetdir output --user demo --master DEMO *.pdf
plop -t output -u demo -m DEMO *.pdf
```

Ein Dokument für ein einziges Empfängerzertifikat verschlüsseln:

```
plop --recipient "certificate={filename=demo_recipient_1.pem}" ↵
        --outfile protected.pdf input.pdf
plop -r "certificate={filename=demo_recipient_1.pem}" -o protected.pdf input.pdf
```

Eine unsichtbare Signatur für ein PDF-Dokument mit Hilfe einer digitalen ID aus der Datei *demo_signer_rsa_2048.p12* erstellen. Das Kennwort für die digitale ID befindet sich in der Datei *pw.txt*:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt" ↵
        --outfile signed.pdf input.pdf
plop -S "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt" ↵
        -o signed.pdf input.pdf
```

Eine Signatur erstellen und diese mit einer manuellen Unterschrift aus einem vorhandenen PDF-Dokument visualisieren:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
        field={rect={100 100 300 adapt}}" --visdoc signature.pdf ↵
        --outfile signed.pdf input.pdf
plop -S "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
        field={rect={100 100 300 adapt}}" --visdoc signature.pdf -o signed.pdf input.pdf
```

4 Sprachbindungen der PLOP- und PLOP DS-Bibliothek

In diesem Kapitel werden die sprachspezifischen Aspekte der PLOP- und PLOP DS-Bibliothek erläutert.

4.1 C-Sprachbindung

PLOP ist in C geschrieben, mit einigen C++-Modulen. Um die C-Sprachbindung zu nutzen, können Sie eine statische oder eine dynamische Bibliothek (DLL/SO) verwenden. Außerdem benötigen Sie die zentrale PLOP-Include-Datei *ploplib.h* zur Einbindung in die Quellmodule Ihrer Anwendung. Alternativ dazu kann *ploplibdl.h* eingesetzt werden, um die PLOP-DLL dynamisch zur Laufzeit zu laden.

Hinweis *Anwendungen, die die C-Sprachbindung von PLOP verwenden, müssen mit einem C++-Linker gebunden werden, da die PLOP-Bibliothek einige in C++ implementierte Teile enthält. Die Verwendung eines C-Linkers kann zu offenen externen Verweisen führen, sofern die Anwendung nicht explizit mit den erforderlichen C++-Laufzeitbibliotheken gebunden wird.*

Fehlerbehandlung. Das PLOP-API bietet einen Mechanismus für die Verarbeitung von Exceptions, die von der Bibliothek ausgelöst werden, um das Fehlen einer integrierten Verarbeitung von Exceptions in der Programmiersprache C zu kompensieren. Mit den *PLOP_TRY( )*- und *PLOP_CATCH( )*-Makros lässt sich Client-Code so einrichten, dass ein dediziertes Codefragment für Fehlerbehandlung und Cleanup aufgerufen wird, wenn eine Exception auftritt. Diese Makros erzeugen zwei Codeabschnitte: den TRY-Block mit Code, der eine Exception auslösen kann, und den CATCH-Block mit dem Code, der eine Exception verarbeitet. Wenn eine der im TRY-Block aufgerufenen Funktionen eine Exception auslöst, wird das Programm direkt an der ersten Anweisung des CATCH-Blocks fortgesetzt. Folgende Regeln müssen im PLOP-Client-Code beachtet werden:

- *PLOP_TRY( )* und *PLOP_CATCH( )* müssen immer paarweise aufgerufen werden.
- *PLOP_new( )* löst nie eine Exception aus; da ein TRY-Block nur mit einem gültigen PLOP-Objekt-Handle gestartet werden kann, muss *PLOP_new( )* außerhalb eines TRY-Blocks aufgerufen werden.
- *PLOP_delete( )* löst nie eine Exception aus und kann daher immer außerhalb eines TRY-Blocks aufgerufen werden. Es kann auch in einem CATCH-Block aufgerufen werden.
- Besondere Aufmerksamkeit ist beim Umgang mit Variablen erforderlich, die sowohl im TRY-Block als auch im CATCH-Block verwendet werden. Da der Compiler nichts vom Sprung zwischen den Blöcken weiß, erzeugt er in einer solchen Situation unter Umständen fehleranfälligen Code (z.B. durch Optimierung des Variablenzugriffs über Register).
  Zum Glück gibt es eine einfache Regel zur Vermeidung dieses Problems: Variablen, die im TRY- und im CATCH-Block verwendet werden, müssen als *volatile* deklariert werden. Das Schlüsselwort *volatile* signalisiert dem Compiler, keine riskanten Optimierungen mit der Variablen durchzuführen.

- Wird ein TRY-Block verlassen (zum Beispiel mit einer *return*-Anweisung, ohne dass das entsprechende Makro *PLOP_CATCH( )* zur Ausführung kommt), muss der Exception-Mechanismus mit dem Makro *PLOP_EXIT_TRY( )* darüber informiert werden.
- Wie in allen PLOP-Sprachbindungen muss die Dokumentverarbeitung beendet werden, wenn eine Exception ausgelöst wurde.

Das folgende Codefragment veranschaulicht diese Regeln durch eine typische Sequenz für den Umgang mit PLOP-Exceptions im Client-Code (eine vollständiges Beispiel finden Sie im PLOP-Paket):

```
if ((plop = PLOP_new()) == (PLOP *) 0)
{
        printf("out of memory\n");
        return(2);
}
PLOP_TRY(plop)
{
        /* statements that directly or indirectly call API functions */
}
PLOP_CATCH(plop)
{
        printf("Error %d in %s(): %s\n",
            PLOP_get_errnum(plop), PLOP_get_apiname(plop),PLOP_get_errmsg(plop));
}
PLOP_delete(plop);
```

Unicode-Verarbeitung von Name-Strings. Die Programmiersprache C bietet erst in Version C11 native Unicode-Unterstützung. Da diese Version aber noch nicht allgemein verbreitet ist, bietet PLOP/PLOP DS Unicode-Unterstützung auf Basis des traditionellen Datentyps *char*. Einige String-Parameter für Funktionen sind als *Name-Strings* deklariert. Diese werden abhängig vom Parameter *length* und dem Vorhandensein eines BOM am Anfang des Strings verarbeitet. Wenn der Parameter *length* in C ungleich 0 ist, wird der String als UTF-16 interpretiert. Wenn der Parameter *length* gleich 0 ist, wird der String als UTF-8 interpretiert, sofern er mit einem UTF-8-BOM beginnt, oder als EBCDIC-UTF-8, sofern er mit einem EBCDIC-UTF-8-BOM beginnt, oder aber als Encoding *host*, sofern kein BOM gefunden werden konnte (oder *ebcdic* auf EBCDIC-basierten Plattformen).

Unicode-Verarbeitung für Optionslisten. Strings innerhalb von Optionslisten erfordern besondere Beachtung, da sie sich nicht als Unicode-Strings im UTF-16-Format ausdrücken lassen, sondern nur als Byte-Arrays. Daher wird für Unicode-Optionen UTF-8 verwendet. Ein BOM am Anfang einer Option legt fest, wie sie zu interpretieren ist. Anhand des BOM wird das Format des Strings bestimmt. Genauer gesagt, wird eine String-Option wie folgt interpretiert:

- Wenn die Option mit einem UTF-8-BOM beginnt *(\xEF\xBB\xBF)*, wird sie als UTF-8 interpretiert.
- Beginnt die Option mit einem EBCDIC-UTF-8-BOM *(\x57\x8B\xAB)*, wird sie als EBCDIC UTF-8 interpretiert.
- Wird kein BOM gefunden, wird der String als *winansi* interpretiert (oder als *ebcdic* auf EBCDIC-Plattformen).

Hinweis *Mit der Hilfsfunktion PLOP_convert_to_unicode( ) lassen sich aus UTF-16-Strings UTF-8-Strings erzeugen, was für die Erzeugung von Optionslisten mit Unicode-Werten nützlich ist.*

Einsatz von PLOP als DLL, die zur Laufzeit geladen wird. Die meisten Clients werden PLOP als statisch gebundene oder dynamische Bibliothek einsetzen, die beim Linken gebunden wird. Sie können die DLL aber auch zur Laufzeit laden und sich dynamisch Pointer auf alle API-Funktionen besorgen. Dies ist besonders nützlich, um die DLL nur bei Bedarf zu laden. Zur einfacheren Verwendung dieser Methode gehen Sie wie folgt vor:

- Binden Sie *ploplibdl.h* statt *ploplib.h* ein.
- Verwenden Sie *PLOP_new_dl( )* und *PLOP_delete_dl( )* statt *PLOP_new( )* und *PLOP_delete( )*.
- Verwenden Sie *PLOP_TRY_DL( )* und *PLOP_CATCH_DL( )* statt *PLOP_TRY( )* und *PLOP_CATCH( )*.
- Arbeiten Sie bei allen anderen PLOP-Aufrufen mit Funktionspointern.
- Kompilieren Sie zusätzlich das Hilfsmodul *ploplibdl.c* und binden Sie die entsprechende Objektdatei mit Ihrer Anwendung.

Das dynamische Laden wird im Beispiel *encryptdl.c* veranschaulicht.

Hinweis *Das Laden der DLL zur Laufzeit wird nicht auf allen Plattformen unterstützt.*

4.2 C++-Sprachbindung

Hinweis *Für in C++ geschriebene .NET-Anwendungen empfehlen wir, auf die .NET-DLL von PLOP direkt zuzugreifen, anstatt über die C++-Sprachbindung (plattformübergreifende Anwendungen sollten allerdings die C++-Sprachbindung verwenden). Das PLOP-Paket enthält C++-Beispielcode für .NET CLI (Common Language Infrastructure), der diese Kombination veranschaulicht.*

Neben der C-Include-Datei *ploplib.h* wird für PLOP-Clients ein objektorientierter Wrapper für C++ mitgeliefert. Dieser erfordert die Include-Datei *plop.hpp*, die wiederum *ploplib.h* inkludiert. Da die Implementierung von *plop.hpp* auf Templates basiert, wird kein entsprechendes *plop.cpp*-Modul benötigt. Der C++-Wrapper ersetzt den funktionalen Ansatz (mit Funktionen und Präfix *PLOP_* in allen PLOP-Funktionsnamen) durch einen eher objektorientierten Ansatz.

String-Behandlung in C++. Mit PLOP 4.1 wurde eine neue Unicode-fähige C++-Sprachbindung eingeführt. Mit dem neuen Template-basierten Ansatz wird die String-Behandlung folgendermaßen unterstützt:

- Strings vom Typ *std::wstring* der C++-Standard-Bibliothek werden als grundlegender String-Typ verwendet. Sie können UTF-16- oder UTF-32-kodierte Unicode-Zeichen enthalten. Dies ist das voreingestellte Verhalten ab PLOP 4.1 und die empfohlene Vorgehensweise für neue Anwendungen, es sei denn, benutzerdefinierte Datentypen bieten einen erheblichen Vorteil gegenüber *wstrings* (siehe nächster Punkt).
- Benutzerdefinierte Datentypen können für die String-Verarbeitung verwendet werden, sofern der benutzerdefinierte Datentyp eine Instantiierung des Klassen-Templates *basic_string* ist und mit vom Client übergebenen Konvertierungsmethoden von und nach Unicode konvertiert werden kann.

Die Standard-Schnittstelle geht davon aus, dass alle an PLOP-Methoden übergebenen und von PLOP-Methoden zurückgegeben Strings native *wstrings* sind. Abhängig von der Größe des Datentyps *wchar_t* müssen *wstrings* Unicode-Strings enthalten, die als UTF-16 (2-Byte-Zeichen) oder UTF-32 (4-Byte-Zeichen) kodiert sind. Literalen Strings im Quellcode muss ein *L* vorangestellt werden, um sie als Wide Strings zu kennzeichnen. Unicode-Zeichen in Literalen können mit der Syntax *\u* und *\U* erstellt werden. Obwohl diese Syntax Teil der ISO-Norm von C++ ist, wird sie von einigen Compilern nicht unterstützt. In diesem Fall müssen literale Unicode-Zeichen mit Hex-Ziffern erstellt werden.

Hinweis *Auf EBCDIC-basierten Systemen erfordert das Formatieren der Optionslisten-Strings für die auf* `wstring` *basierende Schnittstelle eine zusätzliche Konvertierung, um eine Mischung aus EBCDIC- und UTF-16-*`wstrings` *in Optionslisten zu vermeiden. Beispiel-Code sowie Anweisungen für diese Konvertierung finden Sie im Hilfsmodul* `utf16num_ebcdic.hpp`.

Fehlerbehandlung in C++. PLOP-API-Funktionen lösen im Fehlerfall eine C++-Exception aus. Diese Exceptions müssen im Client-Code mit den üblichen *try/catch*-Anweisungen von C++ abgefangen werden. Für ausführlichere Fehlerinformationen stellt die Klasse PLOP die öffentliche Klasse *PLOP::Exception* mit Methoden zur Verfügung, die die genaue Fehlermeldung, die Exception-Nummer sowie den Namen der API-Funktion liefern, die die Exception ausgelöst hat.

Native C++-Exceptions, die durch PLOP-Routinen ausgelöst wurden, verhalten sich wie erwartet. Das folgende Codefragment fängt Exceptions ab, die von PLOP ausgelöst werden:

```
try {
        ...PLOP-Anweisungen...
} catch (PLOP::Exception &ex) {
            wcerr << L"Error " << ex.get_errnum()
            << L" in " << ex.get_apiname()
            << L"(): " << ex.get_errmsg() << endl;
}
```

Einsatz von PLOP als DLL, die zur Laufzeit geladen wird. Ähnlich wie bei der C-Sprachbindung kann PLOP mit der C++-Sprachbindung dynamisch zur Laufzeit an Ihre Anwendung gebunden werden (siehe »Einsatz von PLOP als DLL, die zur Laufzeit geladen wird«, Seite 47). Das dynamische Laden beim Kompilieren des Anwendungsmoduls, das *plop.hpp* inkludiert, können Sie folgendermaßen aktivieren:

```
#define PLOPCPP_DL    1
```

Kompilieren Sie zusätzlich das Hilfsmodul *ploplibdl.c* und binden Sie die entsprechende Objektdatei mit Ihrer Anwendung. Da die Details des dynamischen Ladens im PLOP-Objekt versteckt sind, ist das C++-API davon nicht betroffen: alle Methodenaufrufe sehen gleich aus, unabhängig davon, ob dynamisches Laden aktiviert ist.

Hinweis *Das Laden der DLL zur Laufzeit wird nicht auf allen Plattformen unterstützt.*

4.3 COM-Sprachbindung

Installation der PLOP-Edition für COM. Installieren Sie PLOP/PLOP DS mit der Windows-Installationsroutine. Die Installationsroutine erstellt die notwendigen Registry-Einträge und registriert die PLOP-Komponente bei Windows, so dass sie von jedem COM-kompatiblen Programm verwendet werden kann.

Verarbeitung von Exceptions in COM. Die Verarbeitung von Exceptions für die PLOP-/PLOP DS-Komponente erfolgt entsprechend der COM-Konventionen. Es wird eine COM-Exception mit einer entsprechenden Meldung ausgelöst. Die Exception kann im PLOP-Client mit den üblichen Verfahren abgefangen und bearbeitet werden.

Einsatz der COM-Edition von PLOP mit .NET. Alternativ zu PLOP.NET (siehe Abschnitt 4.5, ».NET-Sprachbindung«, Seite 53) kann die COM-Edition von PLOP auch mit .NET verwendet werden. Dazu müssen Sie aus der COM-Edition von PLOP mit dem Hilfsprogramm *tlbimp.exe* zunächst eine .NET-Assembly erstellen:

```
tlbimp plop_com.dll /namespace:plop_com /out:Interop.plop_com.dll
```

Diese Assembly verwenden Sie dann in Ihrer .NET-Anwendung. Wenn Sie innerhalb von Visual Studio .NET eine Referenz auf *plop_com.dll* hinzufügen, wird automatisch eine Assembly erzeugt.

Das folgende Codefragment zeigt den Einsatz der COM-Edition von PLOP mit C#:

```
Imports plop_com
    ...
Dim p As plop_com.IPDF
    ...
p = New PLOP()
    ...
buf = p.get_buffer()
```

Das folgende Codefragment zeigt den Einsatz der COM-Edition von PLOP mit C#:

```
using plop_com;
    ...
static plop_com.IPDF p;
    ...
p = New PLOP();
    ...
buf = (byte[])p.get_buffer();
```

Der übrige Code ist der gleiche wie bei der .NET-Edition von PLOP. Beachten Sie aber, dass Sie in C# das Ergebnis von *get_buffer( )* konvertieren müssen, da der vom COM-Objekt zurückgegebene VARIANT-Datentyp nicht automatisch konvertiert wird.

4.4 Java-Sprachbindung

Installation der Java-Edition von PLOP. PLOP/PLOP DS ist als native C-Bibliothek implementiert, die über das JNI (Java Native Interface) an Java angebunden ist. Das JDK, das Sie zur Erstellung von Java-Anwendungen verwenden, bietet auch JNI-Unterstützung. Damit die Java-Sprachbindung für PLOP funktioniert, benötigt die Java-VM Zugriff auf die JNI-Bibliothek von PLOP und das PLOP-Java-Paket.

Das Java-Paket von PLOP. Um eine einheitliche Handhabung bei der Java-Programmierung zu gewährleisten, ist PLOP in einem Java-Paket mit folgendem Paketnamen enthalten:

```
com.pdflib.plop
```

Das Java-Paket von PLOP befindet sich in der Datei *plop.jar* und enthält nur die Klasse *plop*. Aktuelle Hinweise zum Einsatz von PLOP in verschiedenen Java-Entwicklungsumgebungen finden Sie in der Datei *readme.txt*.

Um dieses Paket Ihrer Anwendung verfügbar zu machen, müssen Sie *plop.jar* an die Umgebungsvariable *CLASSPATH* anfügen, die Option *-classpath plop.jar* in die Aufrufe von Java-Compiler und Java-Laufzeitumgebung aufnehmen oder die entsprechenden Schritte in Ihrer IDE durchführen. Sie können die Java-VM so konfigurieren, dass sie ein vorgegebenes Verzeichnis nach nativen Bibliotheken durchsucht. Dazu weisen Sie der Property *java.library.path* den Namen des gewünschten Verzeichnisses zu, zum Beispiel:

```
java -Djava.library.path=. encrypt
```

Der Wert dieser Property lässt sich wie folgt überprüfen:

```
System.out.println(System.getProperty("java.library.path"));
```

Außerdem sind die folgenden plattformabhängigen Schritte durchzuführen:

- Unix: Die Bibliothek *libplop_java.so* muss in eines der Standardverzeichnisse für dynamisch ladbare Bibliotheken oder in ein entsprechend konfiguriertes Verzeichnis kopiert werden.
- OS X/macOS: Die Bibliothek *libplop_java.jnilib* muss in eines der Standardverzeichnisse für dynamische Bibliotheken oder in ein entsprechend konfiguriertes Verzeichnis kopiert werden.
- Windows: Die Bibliothek *plop_java.dll* muss ins Windows-Systemverzeichnis oder in ein Verzeichnis kopiert werden, das in der Umgebungsvariablen PATH aufgeführt ist.

PLOP-Servlets und Java-Applikationsserver. PLOP/PLOP DS eignet sich hervorragend für serverseitige Java-Anwendungen, insbesondere für Servlets. Beim Einsatz von PLOP mit einer bestimmten Servlet-Engine sind folgende Konfigurationsaspekte zu beachten:

- Das Verzeichnis, in dem der Servlet-Engine die nativen Bibliotheken erwartet, ist je nach Anbieter unterschiedlich. Üblich sind Systemverzeichnisse, Verzeichnisse der zugrunde liegenden Java-VM oder lokale Servlet-Engine-Verzeichnisse. Einzelheiten hierzu finden Sie in der Dokumentation, die vom Hersteller der Servlet-Engine bereitgestellt wird.

- Servlet-Container verwenden oft einen speziellen Classloader, der möglicherweise Einschränkungen unterliegt oder einen bestimmten Klassenpfad verwendet. Bei manchen Servlet-Engines muss ein besonderer Engine-Klassenpfad festgelegt werden, damit das PLOP-Paket gefunden werden kann.

Beispiele zum Einsatz von PLOP in Servlets finden Sie in der PLOP-Distribution.

Verarbeitung von Exceptions in Java. Alle PLOP/PLOP DS-Methoden lösen im Fehlerfall eine Exception vom Typ *PLOPException* aus. Als PLOP-Benutzer können Sie Exceptions mit den üblichen javaspezifischen Verfahren abfangen und bearbeiten:

```
try {
        plop plop;
        /* ... PLOP-Anweisungen... */

} catch (PLOPException e) {
        System.err.println("encrypt: PLOP Exception occurred:");
        System.err.println(e.get_apiname() +": " + e.get_errmsg());
} finally {
        /* Löschen des PLOP-Objekts */
        if (plop != null) plop.delete();
}
```

4.5 .NET-Sprachbindung

Hinweis *Ausführliche Informationen zu den verschiedenen Ausprägungen und Möglichkeiten für die Verwendung von PLOP mit dem .NET-Framework finden Sie im Dokument PDFlib-in-.NET-HowTo.pdf, das in den Produktpaketen enthalten und auch über die PDFlib-Website verfügbar ist.*

Die .NET-Edition von PLOP unterstützt alle wesentlichen .NET-Konzepte. Technisch gesehen handelt es sich bei der .NET-Edition von PLOP um eine C++-Klasse (mit einem *managed* Wrapper um die *unmanaged* PLOP-Kernbibliothek), die unter Kontrolle des .NET-Frameworks abläuft. Diese Klasse wird als statische Assembly mit einem starken Namen *(strong name)* ausgeliefert. Die PLOP-Assembly *(PLOP_dotnet.dll)* enthält die Bibliothek selbst sowie zusätzliche Meta-Informationen.

Installation der PLOP-Edition für .NET. Installieren Sie PLOP mit der bereitgestellten MSI-Installationsroutine. Die MSI-Installationsroutine von PLOP.NET installiert die PLOP-Assembly einschließlich der zugehörigen Hilfsdateien, Dokumentation und Beispiele interaktiv auf dem Rechner. Außerdem wird PLOP registriert, so dass Sie auf der Registerkarte .NET im Dialogfeld *Add Reference* von Visual Studio .NET sofort darauf zugreifen können.

Fehlerbehandlung in .NET. PLOP.NET unterstützt .NET-Exceptions und löst eine Exception mit detaillierter Fehlermeldung aus, sobald ein Laufzeitproblem auftritt. Der Client ist für das Abfangen der Exception und eine angemessene Reaktion zuständig. Andernfalls fängt das .NET-Framework die Exception ab, was gewöhnlich zum Abbruch der Anwendung führt.

Um Informationen über die Exception zu übermitteln, definiert PLOP eine eigene Exception-Klasse namens *PLOP_dotnet.PLOPException* mit den Members *get_errnum*, *get_errmsg* und *get_apiname*.

Einsatz von PLOP mit C++ und CLI. In C++ geschriebene .NET-Anwendungen (basierend auf der *Common Language Infrastructure* CLI) können ohne die C++-Sprachbindung von PLOP direkt auf die PLOP.NET-DLL zugreifen. Dazu muss PLOP im Quellcode folgendermaßen referenziert werden:

```
using namespace PLOP_dotnet;
```

4.6 Objective-C-Sprachbindung

Obwohl die C- und C++-Sprachbindungen mit Objective-C verwendet werden können, bieten wir auch eine genuine Sprachbindung für Objective-C an. Das PLOP-Framework ist in den folgenden Ausprägungen erhältlich:

- *PLOP* für OS X/macOS
- *PLOP_ios* für iOS

Beide Frameworks enthalten Sprachbindungen für C, C++ und Objective-C.

Installation der Objective-C-Edition von PLOP unter OS X/macOS. Um PLOP in Ihrer Anwendung einsetzen zu können, kopieren Sie *PLOP.framework* oder *PLOP_ios.framework* in das Verzeichnis */Library/Frameworks*. Sie können das PLOP-Framework auch an einem anderen Ort installieren, benötigen dazu aber das *install_name_tool* von Apple, das hier nicht beschrieben wird. Die Header-Datei *PLOP_objc.h* mit den Methoden-Deklarationen von PLOP müssen Sie im Quellcode Ihrer Anwendung importieren:

```
#import "PLOP/PLOP_objc.h"
```

oder

```
#import "PLOP_ios/PLOP_objc.h"
```

Namenskonventionen für Parameter. Für PLOP-Methodenaufrufe müssen Sie die Parameter gemäß der folgenden Konventionen übergeben:

- Der Wert des ersten Parameters wird direkt nach dem Methodennamen, durch einen Doppelpunkt getrennt, angegeben.
- Für jeden weiteren Parameter muss der Parametername mit seinem Wert (wiederum jeweils getrennt durch einen Doppelpunkt) angegeben werden. Die Parameternamen finden Sie in Kapitel 8, »API-Referenz für die PLOP- und PLOP DS-Bibliothek«, Seite 139 und in der Datei *PLOP_objc.h*.

Die folgende Zeile aus der API-Funktionsbeschreibung:

```
int open_document(wstring filename, wstring optlist)
```

entspricht der folgenden Objective-C-Methode:

```
- (NSInteger) open_document: (NSString *) filename optlist: (NSString *) optlist;
```

Ihre Anwendung muss daher ungefähr folgenden Aufruf absetzen:

```
doc = [plop open_document:filename optlist:pageoptlist];
```

Xcode Code Sense kann zur Code-Vervollständigung mit dem PLOP-Framework verwendet werden.

Fehlerbehandlung in Objective-C. Die Objective-C-Sprachbindung übersetzt PLOP-Exceptions in native Objective-C-Exceptions. Bei einem Laufzeitproblem löst PLOP eine native Objective-C-Exception der Klasse *PLOPException* aus. Diese Exceptions können mit der üblichen *try/catch*-Methode behandelt werden:

```
@try {
        ...PLOP-Anweisungen...
}
@catch (PLOPException *ex) {
        NSString * errorMessage =
            [NSString stringWithFormat:@"PLOP error %d in '%@': %@",
            [ex get_errnum], [ex get_apiname], [ex get_errmsg]];
        NSAlert *alert = [[NSAlert alloc] init];
        [alert setMessageText: errorMessage];
        [alert runModal];
        [alert release];
}
@catch (NSException *ex) {
        NSAlert *alert = [[NSAlert alloc] init];
        [alert setMessageText: [ex reason]];
        [alert runModal];
        [alert release];
}
@finally {
        [plop release];
}
```

Außer der Methode *get_errmsg* können Sie auch das Feld *reason* des Exception-Objekts verwenden, um Fehlermeldungen zu erhalten.

4.7 Perl-Sprachbindung

Der PLOP-Wrapper für Perl besteht aus einer C-Wrapperdatei und zwei Perl-Paketmodulen, eins zur Bereitstellung eines Perl-Äquivalents für jede PLOP-API-Funktion und ein weiteres für das PLOP-Objekt. Das C-Modul wird zum Aufbau einer dynamischen Bibliothek verwendet, die vom Perl-Interpreter unter Zuhilfenahme der Paketdatei zur Laufzeit geladen wird. Perl-Skripte referenzieren das Bibliotheksmodul mit einer *use*-Anweisung.

Installation der Perl-Edition von PLOP. Der Erweiterungsmechanismus von Perl lädt dynamische Bibliotheken zur Laufzeit mittels des DynaLoader-Moduls. Perl selbst muss mit Unterstützung dynamischer Bibliotheken kompiliert worden sein (das ist bei den meisten Perl-Konfigurationen der Fall).

Damit die PLOP-Sprachbindung funktioniert, benötigt der Perl-Interpreter Zugriff auf den PLOP-Perl-Wrapper und die Module *plop_pl.pm* und *PDFlib/PLOP.pm*. Zusätzlich zu den unten beschriebenen plattformspezifischen Methoden können Sie mit der Perl-Kommandozeilenoption *-I* ein Verzeichnis zum Modulsuchpfad *@INC* hinzufügen, zum Beispiel:

```
perl -I/path/to/plop encrypt.pl
```

Unix. Perl sucht *plop_pl.so* (unter OS X/macOS: *plop_pl.bundle), plop_pl.pm* und *PDFlib/PLOP.pm* im aktuellen Verzeichnis oder in dem Verzeichnis, das mit folgendem Befehl ausgegeben wird:

```
perl -e 'use Config; print $Config{sitearchexp};'
```

Perl durchsucht außerdem das Unterverzeichnis *auto/plop_pl*. Der obige Befehl liefert eine Ausgabe, die in etwa wie folgt aussieht:

```
/usr/lib/perl5/site_perl/5.16/i686-linux
```

Windows. Die DLL *plop_pl.dll* und die Module *plop_pl.pm* und *PDFlib/PLOP.pm* werden im aktuellen Verzeichnis gesucht oder im Verzeichnis, das mit folgendem Perl-Befehl ausgegeben wird:

```
perl -e "use Config; print $Config{sitearchexp};"
```

Der obige Befehl liefert eine Ausgabe, die in etwa wie folgt aussieht:

```
C:\Programme\Perl5.16\site\lib
```

Verarbeitung von Exceptions in Perl. Wenn eine PLOP-Exception auftritt, wird eine Perl-Exception ausgelöst. Sie kann mit einer *eval*-Sequenz abgefangen und verarbeitet werden:

```
eval {
        ...PLOP-Anweisungen...
};
die "Exception caught: $@" if $@;
```

4.8 PHP-Sprachbindung

Hinweis *Ausführliche Informationen zu den verschiedenen Ausprägungen und Möglichkeiten für die Verwendung von PLOP mit dem PHP finden Sie im Dokument PDFlib-in-.NET-HowTo.pdf, das in den Produktpaketen enthalten und auch über die PDFlib-Website verfügbar ist. Obwohl es vor allem den Einsatz von PDFlib mit PHP betrifft, gilt die Diskussion auch für den Einsatz von PLOP mit PHP.*

Installation der PLOP-Edition für PHP. PLOP/PLOP DS ist als C-Bibliothek implementiert, die dynamisch in PHP eingebunden werden kann. PLOP unterstützt verschiedene PHP-Versionen. Abhängig von der verwendeten PHP-Version müssen Sie die entsprechende PLOP-Bibliothek aus dem entpackten PLOP-Archiv auswählen.

Sie müssen PHP per Konfiguration über die externe PLOP-Bibliothek informieren. Dazu gibt es zwei Möglichkeiten:

- Fügen Sie in *php.ini* eine der folgenden Zeilen ein:

```
extension=plop_php.so           ; für Unix und OS X/macOS
extension=plop_php.dll          ; für Windows
```

  PHP sucht die Bibliothek in dem Verzeichnis, das unter Unix in der Variablen *extension_dir* in der Datei *php.ini* verzeichnet ist. Unter Windows werden außerdem die Standardsystemverzeichnisse durchsucht. Mit dem folgenden einzeiligen PHP-Skript können Sie ermitteln, welche Version der PHP-Sprachbindung von PLOP Sie installiert haben:

```
<?phpinfo()?>
```

  Angezeigt wird eine lange Info-Seite über Ihre aktuelle PHP-Konfiguration. Suchen Sie auf der Seite nach dem Abschnitt *plop*. Wenn dieser Abschnitt den Satz enthält

```
PDFlib PLOP (PDF Linearization, Optimization, Protection and Digital Signature) =>
enabled
```

  (plus der PLOP-Versionsnummer), haben Sie PLOP für PHP erfolgreich installiert.
- Laden Sie PLOP zur Laufzeit, wobei Sie eine der folgenden Zeilen an den Anfang Ihres Skripts stellen müssen:

```
dl("plop_php.so");              # für Unix und OS X/macOS
dl("plop_php.dll");             # für Windows
```

Behandlung von Dateinamen in PHP. Nicht qualifizierte Dateinamen (also solche ohne jede Pfadangabe) sowie relative Dateinamen für PDF-, Rasterbild-, Font- und andere Dateien auf dem Laufwerk werden in der Unix- und der Windows-Version von PHP unterschiedlich behandelt:

- Unter Unix sucht PHP Dateien ohne Pfadangabe in dem Verzeichnis, in dem sich das Skript befindet.
- Unter Windows sucht PHP Dateien ohne Pfadangabe nur in dem Verzeichnis, in dem sich die PHP-DLL befindet.

Verarbeitung von Exceptions in PHP. Da PHP strukturierte Ausnahmebehandlung unterstützt, werden PLOP-Exceptions als PHP-Exceptions weitergegeben. PLOP-Exceptions können also mit der üblichen Kombination aus *try/catch* abgefangen werden:

```
try {

...PLOP-Anweisungen...

} catch (PLOPException $e) {
    print "PLOP exception occurred:\n";
    print "[" . $e->get_errnum() . "] " . $e->get_apiname() . ": "
                $e->get_errmsg() . "\n";
}
catch (Exception $e) {
    print $e;
}
```

Entwicklung mit Eclipse und Zend Studio. Die PHP Development Tools (PDT) unterstützen die PHP-Entwicklung mit Eclipse und Zend Studio. Für PDT kann kontextsensitive Hilfe mit den unten beschriebenen Schritten konfiguriert werden.

Fügen Sie PLOP zu den Eclipse-Voreinstellungen hinzu, um es bei allen PHP-Projekten bekannt zu machen:

- Wählen Sie *Window, Preferences, PHP, PHP Libraries, New...* Ein Assistent wird gestartet.
- Fügen Sie unter *User library name* das Wort *PLOP* ein, klicken Sie auf *Add External folder...* und wählen Sie das Verzeichnis *bind\php\Eclipse PDT*.

In einem bestehenden oder neuen PHP-Projekt können Sie folgendermaßen einen Verweis auf die PLOP-Bibliothek hinzufügen:

- Klicken Sie im PHP-Explorer mit der rechten Maustaste auf das PHP-Projekt und wählen Sie *Include Path, Configure Include Path...*
- Gehen Sie zur Registermarke *Libraries*, klicken Sie auf *Add Library...* und wählen Sie *User Library*, *PLOP*.

Dann können Sie in der PHP-Explorer-Ansicht die Liste der PLOP-Methoden unter dem Knoten *PHP Include Path/PLOP/PLOP* durchsuchen. Beim Schreiben von neuem PHP-Code bietet Eclipse mit Code-Vervollständigung und kontextsensitiver Hilfe Unterstützung für alle PDFlib-Methoden.

4.9 Python-Sprachbindung

Installation der Python-Edition von PLOP. Der Erweiterungsmechanismus von Python lädt dynamische Bibliotheken zur Laufzeit. Damit die PLOP-Sprachbindung funktioniert, benötigt der Python-Interpreter Zugriff auf die PLOP-Bibliothek für Python, nach der in den Verzeichnissen gesucht wird, die in der Umgebungsvariable PYTHONPATH aufgeführt sind. Der Name des Python-Wrappers ist plattformabhängig:

- Unix und OS X/macOS: *plop_py.so*
- Windows: *plop_py.pyd*

Fehlerbehandlung in Python. Die Python-Sprachbindung installiert einen speziellen Error-Handler, der PLOP-Fehler in native Python-Exceptions übersetzt. Die Python-Exceptions können mit der üblichen Kombination aus TRY und EXCEPT behandelt werden:

```
try:
        ...PLOP-Anweisungen...
except PLOPException:
        print 'PLOP Exception caught!'
```

4.10 Ruby-Sprachbindung

Installation der Ruby-Edition von PLOP. Der Erweiterungsmechanismus von Ruby lädt eine dynamische Bibliothek zur Laufzeit. Damit die PLOP-Sprachbindung funktioniert, benötigt der Ruby-Interpreter Zugriff auf die PLOP-Erweiterungsbibliothek für Ruby. Diese Bibliothek (unter Windows und Unix: *PLOP.so*; unter OS X/macOS: *PLOP.bundle)* wird normalerweise im Unterverzeichnis *site_ruby* des lokalen Ruby-Installationsverzeichnisses installiert, das heißt in einem Verzeichnis mit etwa folgendem Namen:

```
/usr/local/lib/ruby/site_ruby/<version>/
```

Ruby durchsucht aber auch andere Verzeichnisse nach Erweiterungen. Mit folgendem Ruby-Aufruf erhalten Sie eine Liste dieser Verzeichnisse:

```
ruby -e "puts $:"
```

Diese Liste enthält in der Regel auch das aktuelle Verzeichnis, so dass Sie die PLOP-Erweiterungsbibliothek und die Skripten zum Testen einfach ins gleiche Verzeichnis stellen können.

Fehlerbehandlung in Ruby. Die Ruby-Sprachbindung installiert einen Error-Handler, der PLOP-Exceptions in native Ruby-Exceptions übersetzt. Die Ruby-Exceptions können mit der üblichen *rescue*-Technik verarbeitet werden:

```
begin
        ...PLOP-Anweisungen...
rescue PLOPException => pe
  print "PLOP exception occurred in encrypt sample:\n"
  print "[" + pe.get_errnum.to_s + "] " + pe.get_apiname + ": " + pe.get_errmsg + "\n"
end
```

5 Kennwortschutz

5.1 Kennwortschutz in PDF

PDF-Kennwortschutz bietet folgende Sicherheitsfunktionen:

- Das Benutzerkennwort (auch Kennwort zum Öffnen des Dokuments) ist zum Öffnen der Datei erforderlich.
- Das Master-Kennwort (auch Berechtigungskennwort) ist zum Ändern von Sicherheitseinstellungen wie Berechtigungen, Benutzer- oder Master-Kennwort erforderlich. Dateien mit Benutzer- und Master-Kennwort können mit einem von beiden Kennwörtern geöffnet werden.
- Berechtigungen beschränken die mit dem PDF-Dokument erlaubten Aktionen, wie zum Beispiel das Drucken oder das Extrahieren von Text.
- Dateianhänge können separat verschlüsselt werden, ohne dass das Dokument selbst verschlüsselt ist.

Verwendet ein PDF-Dokument eine dieser Sicherheitsfunktionen, wird es verschlüsselt. Zum Anzeigen oder Ändern der Sicherheitseinstellungen eines Dokuments klicken Sie

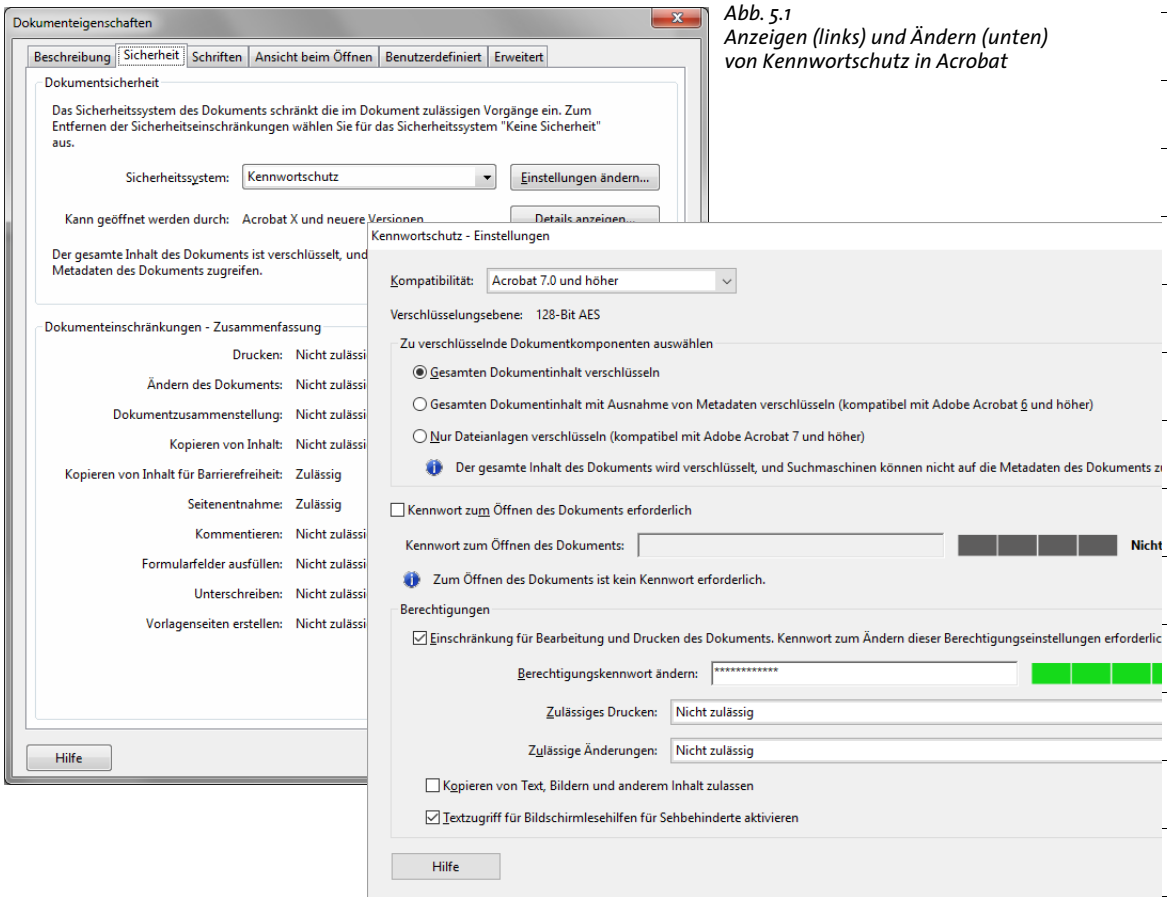


Abb. 5.1
Anzeigen (links) und Ändern (unten) von Kennwortschutz in Acrobat

in Acrobat auf *Datei, Eigenschaften, Sicherheit, Details anzeigen* bzw. *Einstellungen ändern*. Abbildung 5.1 zeigt Dialogfenster für die Sicherheitseinstellungen von Acrobat.

Verschlüsselungsalgorithmen und Schlüssellängen. Bei der PDF-Verschlüsselung werden folgende Verschlüsselungsalgorithmen verwendet:

- RC4, eine symmetrische Stromverschlüsselung (derselbe Algorithmus kann zum Verschlüsseln wie zum Entschlüsseln verwendet werden). RC4 bietet keinen adäquaten Schutz mehr und wurde in PDF 2.0 als veraltet deklariert.
- AES *(Advanced Encryption Standard)* wurde im Standard FIPS-197 spezifiziert. AES ist eine moderne Block-Verschlüsselung, die in einer Vielzahl von Anwendungen verwendet wird.

Da die eigentlichen Schlüssel unhandliche binäre Sequenzen sind, werden sie von benutzerfreundlichen Kennwörtern abgeleitet, die aus normalem Text bestehen. Im Zuge der PDF- und Acrobat-Entwicklung wurden die PDF-Verschlüsselungsmethoden erweitert, um stärkere Algorithmen, längere Schlüssel und komplexere Kennwörter verwenden zu können. Tabelle 5.1 gibt eine Übersicht über die Merkmale von Verschlüsselung, Schlüsseln und Kennwörtern für alle PDF-Versionen.

Tabelle 5.1 Verschlüsselungsalgorithmen, Schlüssellänge und Länge von Kennwörtern für verschiedene PDF-Versionen

PDF- und Acrobat-Version, pCOS Algorithmus-Nummer	***Verschlüsselungsalgorithmus und Schlüssellänge***	***Max. Länge des Kennworts und Kennwort-Encoding***
PDF 1.1 - 1.3 (Acrobat 2-4), pCOS-Algorithmus 1	*RC4 40-Bit (schwacher Algorithmus, in PDF 2.0 als veraltet deklariert)*	*32 Zeichen (Latin-1)*
PDF 1.4 (Acrobat 5), pCOS-Algorithmus 5	*RC4 128-Bit (schwacher Algorithmus, in PDF 2.0 als veraltet deklariert)*	*32 Zeichen (Latin-1)*
PDF 1.5 (Acrobat 6), pCOS-Algorithmus 3	*RC4 128-Bit wie bei PDF 1.4, aber andere Anwendung der Verschlüsselungsmethode (schwacher Algorithmus, in PDF 2.0 als veraltet deklariert)*	*32 Zeichen (Latin-1)*
PDF 1.6 (Acrobat 7) und PDF 1.7 = ISO 32000-1 (Acrobat 8), pCOS-Algorithmus 4	*AES-128 (in PDF 2.0 als veraltet deklariert)*	*32 Zeichen (Latin-1)*
PDF 1.7ext3 (Acrobat 9), pCOS-Algorithmus 9	*AES-256 mit mangelhafter Handhabung von Kennwörtern (schwacher Algorithmus, in PDF 2.0 als veraltet deklariert)*	*127 UTF-8-Bytes (Unicode)*
PDF 1.7ext8 (Acrobat X/XI/DC) und PDF 2.0 = ISO 32000-2, pCOS-Algorithmus 11	*AES-256 mit verbesserter Handhabung von Kennwörtern*	*127 UTF-8-Bytes (Unicode)*

Beachten Sie, dass AES-256 von nativen PDF-Viewern unter OS X/macOS (getestet bis OS X 10.10.3) und iOS (getestet bis iOS 8.1.3) nicht unterstützt wird.

Kennwörter. Die PDF-Verschlüsselung arbeitet intern je nach PDF-Version mit 40-, 128- oder 256-Bit-Schlüsseln. Der binäre Chiffrierschlüssel wird aus dem vom Benutzer übergebenen Kennwort abgeleitet. Für das Kennwort bestehen Beschränkungen in der Länge und beim Encoding:

- Bis PDF 1.7 (ISO 32000-1) durften Kennwörter maximal 32 Zeichen lang sein und nur Zeichen aus dem Encoding Latin-1 enthalten.

- Mit PDF 1.7ext3 wurden Unicode-Zeichen eingeführt und die maximale Länge in der UTF-8-Darstellung des Kennworts auf 127 Bytes erhöht. Da UTF-8 Zeichen mit einer variablen Länge von 1-4 Bytes kodiert sind, ist die zulässige Anzahl von Unicode-Zeichen im Kennwort kleiner als 127, wenn es Nicht-ASCII-Zeichen enthält. Da japanische Zeichen in der Regel 3 Bytes in der UTF-8-Darstellung benötigen, können bis zu 42 japanische Zeichen für ein Kennwort verwendet werden.

Um Mehrdeutigkeiten zu vermeiden, werden Unicode-Kennwörter durch einen Prozess namens *SASLprep* normalisiert (spezifiziert in RFC 4013, der auf *Stringprep* in RFC 3454 basiert). Dieser Prozess beseitigt Nicht-Textzeichen und normalisiert bestimmte Zeichenklassen (Nicht-ASCII-Leerzeichen werden z.B. auf ASCII-Leerzeichen U+0020 abgebildet). Das Kennwort wird in der Unicode-Normalform NFKC normalisiert. Spezielle Verarbeitungsschritte für bidirektionalen Text sollen Mehrdeutigkeiten vermeiden, wenn etwa links- und rechtsläufige Zeichen in einem Kennwort vermischt werden.

Die Stärke der PDF-Verschlüsselung hängt nicht nur von der Länge des Chiffrierschlüssels ab, sondern auch von der Länge und Qualität des Kennworts. Eigennamen oder echte Wörter sollten bekanntermaßen nicht als Kennwörter verwendet werden, da sie leicht zu erraten sind oder sich mit einem sogenannten Wörterbuchangriff systematisch ausprobieren lassen. Untersuchungen haben ergeben, dass für sehr viele Kennwörter einfach der Name des Partners oder Haustiers, der eigene Geburtstag, der Spitzname des Kindes usw. verwendet werden, die sich leicht erraten lassen.

Berechtigungseinschränkungen. In PDF können verschiedene Berechtigungen für Dokumente individuell vergeben oder eingeschränkt werden (siehe Abbildung 5.1):

- *Drucken zulässig:* Wenn Drucken unzulässig ist, deaktiviert Acrobat die zugehörige Funktion. Acrobat unterscheidet zwischen Drucken in *Geringer Auflösung (150 dpi)* und in *Hoher Auflösung*. Drucken mit niedriger Auflösung erzeugt ein Rasterbild auf der Seite, das sich nur für den persönlichen Gebrauch, aber nicht für hochwertige Reproduktion oder erneutes Destillieren eignet. Beachten Sie, dass Rasterbild-basierter Druck nicht nur zu geringer Ausgabequalität führt, sondern das Drucken auch erheblich verlangsamt.
- *Zulässige Änderungen:* Über die zugehörige Liste kann festgelegt werden, welche Änderungen an dem PDF-Dokument vorgenommen werden dürfen:

  ```
  Einfügen, Löschen und Drehen von Seiten
  Ausfüllen von Formularfeldern und Unterschreiben vorhandener Unterschriftsfelder
  Kommentieren, Ausfüllen von Formularfeldern und Unterschreiben vorhandener
  Unterschriftsfelder
  Alles außer Entnehmen von Seiten
  ```

- Das Kopieren von Inhalten wird über *Kopieren von Text, Bildern und anderen Inhalten zulassen* gesteuert. Dies kann für Barrierefreiheit aktiviert werden über *Textzugriff für Bildschirmlesehilfen für Sehbehinderte aktivieren*. Diese Einstellung gilt jedoch bei PDF 2.0 als veraltet, da ein PDF-Viewer stets barrierefrei funktionieren sollte.

Bei einer Zugriffsbeschränkung für ein Dokument, zum Beispiel mit der Einstellung *Drucken zulässig: Keine* deaktiviert Acrobat die zugehörige Funktion. Dies gilt jedoch nicht unbedingt für PDF-Viewer oder andere Software von Drittherstellern. Es ist Sache der jeweiligen Tool-Entwickler, ob sie die definierten Berechtigungen berücksichtigen oder nicht. Es gibt einige PDF-Tools, die Berechtigungseinstellungen vollständig ignorieren; außerdem können Zugriffsbeschränkungen mit kommerziellen Cracker-Tools

außer Kraft gesetzt werden. Dies ist jedoch vom Schlüsselknacken zu unterscheiden; es ist einfach nicht möglich, das Drucken einer PDF-Datei zuverlässig zu unterbinden, wenn sie am Bildschirm anzeigbar sein soll. Dies wird sogar im ISO-Standard 32000-1 dokumentiert:

»Sobald das Dokument geöffnet und erfolgreich entschlüsselt wurde, hat ein PDF-Anzeigeprogramm technisch gesehen Zugriff auf den gesamten Inhalt des Dokuments. Die PDF-Verschlüsselung enthält keine Mechanismen, die eine Durchsetzung der im Encryption-Dictionary festgelegten Dokumentberechtigungen erzwingen.«

Verschlüsselung einzelner Dokumentkomponenten. Standardmäßig umfasst die PDF-Verschlüsselung immer alle Komponenten eines Dokuments. Allerdings kann es manchmal sinnvoll sein, gezielt nur einzelne Komponenten eines Dokuments zu verschlüsseln:

- Ab PDF 1.5 (Acrobat 6) wurde die Funktion »Metadaten als Klartext« eingeführt. Damit können verschlüsselte Dokumente unverschlüsselte XMP-Metadaten enthalten. Dies erlaubt Suchmaschinen, auch aus verschlüsselten Dokumenten Metadaten auszulesen.
- Ab PDF 1.6 (Acrobat 7) können Dateianhänge auch in ansonsten ungeschützten Dokumenten verschlüsselt werden. Damit kann ein ungeschütztes Dokument als Container für vertrauliche Anhänge dienen.

Sicherheitsempfehlungen. Um höchstmögliche Sicherheit zu erzielen, sollten Sie folgende Punkte vermeiden:

- Vermeiden Sie Kennwörter aus nur 1-6 Zeichen, da sie anfällig sind für Angriffe, bei denen alle möglichen Kennwörter systematisch durchprobiert werden (Brute-Force-Angriff).
- Vermeiden Sie echte Wörter, da sie anfällig sind für Angriffe, bei denen alle möglichen echten Wörter systematisch durchprobiert werden (Wörterbuchangriff). Kennwörter sollten nicht-alphanumerische Zeichen enthalten. Verwenden Sie nicht einfach den Namen Ihres Partners oder Haustiers, den eigenen Geburtstag oder andere leicht zu erratende Begriffe.
- Vermeiden Sie Verschlüsselung mit dem schwachen RC4-Algorithmus und AES-256 gemäß PDF 1.7ext3 (Acrobat 9), da diese aufgrund einer Schwachstelle bei der Kennwortprüfung anfällig für Brute-Force-Angriffe auf Kennwörter ist. Deshalb verwenden Acrobat X/XI/DC und PLOP nie die Verschlüsselung gemäß Acrobat 9 zum Schutz neuer Dokumente (sondern nur zur Entschlüsselung vorhandener Dokumente).

Zusammengefasst: Verwenden Sie am besten AES-256 gemäß PDF1.7 ext8/PDF 2.0. Kennwörter sollten mindestens 6 Zeichen lang sein und auch nicht-alphanumerische Zeichen enthalten.

PDF-Schutz im Web. Wenn PDF-Dokumente über das Web bereitgestellt werden, können Benutzer mit dem Browser immer eine lokale Kopie des Dokuments erstellen. Es gibt keine Möglichkeit für PDF-Dokumente, dies zu verhindern.

5.2 Kennwortschutz für Dokumente mit PLOP einrichten

Mit PLOP können Sie Standardsicherheitsfunktionen auf PDF-Dateien anwenden oder daraus entfernen. PLOP kann PDF-Dokumente mit Benutzer- und Master-Kennwort versehen sowie Zugriffsbeschränkungen festlegen, die z.B. das Drucken des Dokuments in Acrobat, das Kopieren von Text oder das Ändern des Dokuments verhindern. Zur Entschlüsselung eines Dokuments ist das zugehörige Master-Kennwort erforderlich.

Verschlüsselungsalgorithmus und Schlüssellänge für Kennwortschutz. PLOP verwendet immer AES-128 (pCOS-Algorithmus 4) oder die sichere Variante von AES-256 (pCOS-Algorithmus 11). PLOP verwendet weder den schwachen RC4-Algorithmus noch die schwache Variante von AES-256 gemäß PDF 1.7ext3/Acrobat 9 (pCOS-Algorithmus 9), die eine Schwachstelle bei der Verarbeitung des Kennworts hat. Der Verschlüsselungsalgorithmus lässt sich mit der Option *encryption* von *PLOP_create_document( )* auswählen:

- Bei *encryption=algo4:* die PDF-Version wird auf PDF 1.6 erhöht, sofern erforderlich, und AES-128-Verschlüsselung gemäß Acrobat 7/8 (pCOS-Algorithmus 4) wird angewendet. Kennwörter dürfen nur aus dem Latin-1-Encoding bestehen und sind auf 32 Zeichen beschränkt.
- Bei *encryption=algo11:* (Standardwert): die PDF-Version wird auf PDF 1.7ext3 erhöht, sofern erforderlich, und AES-256-Verschlüsselung gemäß Acrobat X/XI/DC (pCOS-Algorithmus 11) wird angewendet. Kennwörter dürfen Unicode-Zeichen enthalten und sind auf 127 UTF-8-Bytes beschränkt.

Erforderliche Kennwörter für PLOP-Operationen. Um die Vorgaben des Verfassers zu respektieren, die in den Berechtigungseinstellungen eines PDF-Dokuments hinterlegt sind, können an kennwortgeschützten Dokumenten unter Umständen nicht alle Operationen durchgeführt werden. PLOP verarbeitet das Dokument dann gemäß folgender Regeln:

- Die Abfrage des Verschlüsselungsstatus mit dem pCOS-Pseudo-Objekt *encrypt/algorithm* usw. unabhängig von jedem Kennwort ist immer erlaubt.
- Die Abfrage von Dokumenteigenschaften über die pCOS-Schnittstelle wird durch den pCOS-Modus geregelt. Zum Beispiel können XMP-Metadaten des Dokuments, Dokument-Infofelder, Lesezeichen und Anmerkungsinhalte ohne das Master-Kennwort abgerufen werden, wenn das Dokument kein Benutzerkennwort verlangt (oder nur das Benutzerkennwort übergeben wurde). Weiterführende Informationen hierzu finden Sie in der pCOS-Pfadreferenz.
- Folgende Operationen erfordern das Master-Kennwort: Ändern oder Entfernen des Benutzer-, Master-Kennworts oder der Berechtigungen, sowie Linearisieren, Optimieren, Reparieren oder Signieren eines verschlüsselten Dokuments.

Tabelle 5.2 zeigt die jeweiligen Voraussetzungen für die verschiedenen Operationen.

Kennwörter mit PLOP setzen. In der PLOP-Bibliothek und im PLOP-Kommandozeilen-Tool nennen wir das PDF-Originaldokument das Eingabedokument *(input)* und das verbzw. entschlüsselte Ergebnis das Ausgabedokument *(output),* auch wenn beide den selben Dateinamen erhalten. Ist das Eingabedokument geschützt, benötigt PLOP je nach gewünschter Operation das Benutzer- oder Master-Kennwort (siehe Tabelle 5.2). Konnte das Eingabedokument erfolgreich geöffnet werden (entweder weil es ungeschützt war

Tabelle 5.2 Erforderliche Kennwörter für verschiedene Operationen an verschlüsselten Dokumenten

bekannte Kennwörter	Verschlüsselungsstatus abfragen (pCOS-Pseudo-Objekt »encrypt«)	Dokument-Info, XMP-Metadaten, Lesezeichen, Anmerkungsinhalte mit pCOS abfragen	Kennwörter oder Berechtigungen ändern, Linearisieren, Optimieren, Reparieren oder Signieren
keine	ja	nur, wenn kein Benutzerkennwort gesetzt ist	nein
Benutzer	ja	ja	nein
Master	ja	ja	ja

oder weil das passende Kennwort übergeben wurde), kann das Ausgabedokument mit einer beliebigen Kombination aus Benutzerkennwort, Master-Kennwort und Berechtigungseinstellungen versehen werden. PLOP verarbeitet die vom Client übergebenen Kennwörter für das erzeugte Dokument auf folgende Weise:

- Wenn ein Benutzerkennwort oder Berechtigungen, aber kein Master-Kennwort übergeben wurde, könnte ein normaler Benutzer die Sicherheitseinstellungen ändern und damit jeglichen Schutz umgehen. Aus diesem Grund behandelt PLOP diese Situation als Fehler.
- Sind Benutzer- und Master-Kennwort identisch, wäre keine Unterscheidung zwischen Benutzer und Eigentümer der Datei mehr möglich, was einem effektiven Schutz zuwiderläuft. PLOP behandelt diese Situation als Fehler.
- Bei AES-256 sind Unicode-Kennwörter erlaubt. Bei allen älteren Verschlüsselungsalgorithmen sind Kennwörter auf den Latin-1-Zeichensatz beschränkt. Wenn diese Regel verletzt wird, wird eine Exception ausgelöst.
- Bei AES-256 werden Kennwörter auf 127 UTF-8-Bytes und bei älteren Verschlüsselungsalgorithmen auf 32 Zeichen gekürzt.

Berechtigungen mit PLOP setzen. Mit PLOP lassen sich Zugriffsberechtigungen abfragen, setzen oder entfernen, wie in Tabelle 5.3 dargestellt. Sofern nicht anders angegeben, sind standardmäßig alle Aktionen erlaubt. Bei der Festlegung von Berechtigungen wird die zugehörige Funktion in Acrobat deaktiviert. Zum Setzen von Berechtigungen ist immer ein Master-Kennwort erforderlich, jedoch kein Benutzerkennwort. Tabelle 5.3 zeigt die unterstützten Schlüsselwörter zum Setzen von Berechtigungen.

Tabelle 5.3 Schlüsselwörter für Berechtigungseinschränkungen für die Option `permissions` *von PLOP_create_document( ) und PLOP_add_recipient( )*

Schlüsselwort	Beschreibung
Drucken des Dokuments	
nohiresprint	*Drucken in höher Auflösung verhindern. Wurde* `noprint` *nicht angegeben, wird der Ausdruck mit der Option »Als Bild drucken« durchgeführt und die Seite in niedriger Auflösung ausgegeben.*
noprint	*Drucken der Datei verhindern.*
Ändern des Dokuments	
nomodify	*Hinzufügen von Formularfeldern oder andere Änderungen verhindern.*
noannots	*Hinzufügen oder Ändern von Kommentaren und Ausfüllen von Formularfeldern verhindern. Wurden* `nomodify` *und* `noannots` *nicht angegeben, ist das Erstellen und Ändern von Formularfeldern (einschließlich Signaturfeldern) erlaubt.*

Tabelle 5.3 Schlüsselwörter für Berechtigungseinschränkungen für die Option `permissions` *von PLOP_create_document() und PLOP_add_recipient()*

Schlüsselwort	***Beschreibung***
noforms	*(Impliziert* `noannots`*) Ausfüllen von Formularfeldern und Signieren verhindern, auch wenn* `noannots` *nicht angegeben wurde.*
noassemble	*(Impliziert* `nomodify`*) Einfügen, Löschen oder Drehen von Seiten und die Erstellung von Lesezeichen und Miniaturansichten (Thumbnails) verhindern, auch wenn* `nomodify` *nicht angegeben wurde.*
Kopieren von Inhalten aus dem Dokument	
nocopy	*Kopieren und Extraktion von Text oder Grafiken.*
noaccessible	*(In PDF 2.0 als veraltet deklariert) Extraktion von Text oder Grafik zum barrierefreien Zugang verhindern.*
weitere	
plainmetadata	*(Nur für PLOP_create_document()) Die XMP-Metadaten des Dokuments bleiben auch bei verschlüsselten Dokumenten unverschlüsselt.*
nomaster	*Nur für PLOP_add_recipient()) Drucken, Bearbeiten und Inhaltsextraktion verhindern gemäß den angegebenen Berechtigungsschlüsselwörtern sowie Änderungen der Dokument-Sicherheitseinstellungen verhindern. Wird dieses Schlüsselwort nicht übergeben, hat der Empfänger die vollen Berechtigungen für das Dokument, d.h. alle weiteren Berechtigungsschlüsselwörter (außer* `plainmetadata`*) werden ignoriert.*

Beachten Sie, dass in Acrobat die vier Berechtigungseinschränkungen für Dokumentänderungen nicht alle separat gesetzt werden können, sondern einige zu Gruppen zusammengefasst sind. Tabelle 5.4 enthält einen Vergleich der Acrobat-Einstellungen (für die Werte von »Zulässige Änderungen« siehe Abbildung 5.1) und den entsprechenden PLOP-Berechtigungsschlüsselwörtern.

Tabelle 5.4 Berechtigungen in Acrobat und entsprechende Schlüsselwortkombinationen für die Option `permissions` *von PLOP_create_document() und PLOP_add_recipient()*

»Zulässige Änderungen« in Acrobat	***zugehörige Schlüsselwörter für Berechtigungen***
Keine	`nomodify noannots noforms noassemble`
Einfügen, Löschen und Drehen von Seiten	`nomodify noannots noforms`
Hinzufügen von Formularfeldern und Unterschreiben vorhandener Unterschriftsfelder	`nomodify noannots noassemble`
Kommentieren, Hinzufügen von Formularfeldern und Unterschreiben vorhandener Unterschriftsfelder	`nomodify noassemble`
Alles außer Entnehmen von Seiten	`noassemble`

5.3 Anwenden von Kennwortschutz auf der Kommandozeile

Mit den Optionen *userpassword* oder *masterpassword* (oder beiden) von *PLOP_create_document( )* können Sie Dokumente verschlüsseln. Beachten Sie dabei, dass ein Benutzerkennwort immer auch ein Master-Kennwort erfordert, aber nicht umgekehrt. Vollständige Codebeispiele zum Verschlüsseln und Entschlüsseln von PDF-Dokumenten und der PLOP-Bibliothek finden Sie in den Programmbeispielen *encrypt* und *decrypt*, die in allen PLOP-Paketen enthalten sind. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Optionen *--user* und *--master*.

Zugriffsberechtigungen können Sie mit der Option *permissions* von *PLOP_create_document( )* setzen; für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--permissions*.

Hinweis *Unter Windows dürfen Kennwörter in Kommandozeilen-Optionen Unicode-Zeichen außerhalb des Latin-1-Encodings enthalten.*

Beispiele für Verschlüsselung. In den Beispielen für Kommandozeilenaufrufe unten werden die langen Optionsnamen verwendet; für die Kurzform der Optionen siehe Abschnitt 3.1, »PLOP- und PLOP DS-Kommandozeilen-Optionen«, Seite 39.

Verschlüsseln einer Datei mit dem Benutzerkennwort *demo* und dem Master-Kennwort *DEMO:*

```
plop --user demo --master DEMO --outfile encrypted.pdf input.pdf
```

Alle Dateien im aktuellen Verzeichnis mit dem selben Benutzerkennwort *demo* und dem Master-Kennwort *DEMO* verschlüsseln und die Ausgabedateien ins Verzeichnis *output* kopieren:

```
plop --targetdir output --user demo --master DEMO *.pdf
```

Kennwörter mit Leerzeichen müssen in geschweifte Klammern (gemäß der Syntax für Optionslisten) und in gerade Anführungszeichen (gemäß der Shell-Syntax) gesetzt werden, wie im folgenden Beispiel gezeigt: Verschlüsseln eines Dokuments mit dem Master-Kennwort *two words*:

```
plop --master "{two words}" --outfile encrypted.pdf input.pdf
```

Verschlüsseln und Signieren eines Dokuments (für die Signaturoptionen siehe Abschnitt 7.2.2, »Signieren mit der integrierten Engine«, Seite 97):

```
plop --user demo --master DEMO --outfile signed+encrypted.pdf ↵
        --signopt "update=false digitalid={filename=demo_signer_rsa_2048.p12} ↵
        password=demo" input.pdf
```

Zugriffsberechtigungen. Das Master-Kennwort *DEMO* und die Zugriffsbeschränkungen *noprint, nocopy* und *noannots* für alle Dateien eines Verzeichnisses setzen und die Ausgabedateien in das Verzeichnis *output* kopieren. Unabhängig von der in den Eingabedokumenten verwendeten Verschlüsselung wird für alle Dateien AES-Verschlüsselung verwendet. Mit dem Verbosity-Level 2 werden die Namen aller Ein- und Ausgabedateien bei der Verarbeitung ausgegeben:

```
plop --verbose 2 --master DEMO ↵
        --permissions "noprint nocopy noannots" --targetdir output *.pdf
```

Alle Zugriffsbeschränkungen aus einer Datei entfernen und das Ergebnis in eine andere Ausgabedatei mit demselben Master-Kennwort kopieren. Dazu wird das Master-Kennwort des Eingabedokuments benötigt:

```
plop --password DEMO --master DEMO --outfile unrestricted.pdf protected.pdf
```

Ein Dokument erneut verschlüsseln (z.B. um schwache Verschlüsselung durch starke AES-Verschlüsselung zu ersetzen oder schwache Kennwörter durch starke) und die Zugriffsberechtigungen des Eingabedokuments klonen. Das Ergebnis in eine andere Ausgabedatei kopieren. Dazu wird das Master-Kennwort des Eingabedokuments benötigt:

```
plop --password DEMO --master LONGPASSWORD --permissions keep ↵
        --outfile unrestricted.pdf protected.pdf
```

Beispiele für Entschlüsselung. Entschlüsseln einer Datei mit dem Master-Kennwort *DEMO*. Alle im Eingabedokument eventuell gesetzten Zugriffsbeschränkungen werden dabei entfernt (da die Ausgabe nicht verschlüsselt ist):

```
plop --password DEMO --outfile decrypted.pdf encrypted.pdf
```

Erneutes Verschlüsseln mit einem stärkeren Verschlüsselungsalgorithmus. Mit PLOP lässt sich der Schutz von Dokumenten erhöhen, die nur kurze Chiffrierschlüssel oder schwache Kennwörter verwenden. Sie müssen das alte und das neue Kennwort übergeben. PLOP verwendet standardmäßig die starke AES-Verschlüsselung. Die folgenden Beispiele gehen davon aus, dass die Eingabe mit dem Master-Kennwort *old* verschlüsselt ist und die Ausgabe mit dem Master-Kennwort *DEMO* verschlüsselt wird. Das neue Kennwort kann sogar mit dem alten identisch sein. Selbstverständlich sollten Sie nur starke Kennwörter und keine schwachen wie in diesem Beispiel verwenden (siehe auch »Sicherheitsempfehlungen«, Seite 64):

```
plop --password old --master DEMO --outfile strong.pdf weak.pdf
```

6 Zertifikatsicherheit

6.1 Zertifikatsicherheit in Acrobat

Vorteile von Zertifikatsicherheit. Mit Kennwortschutz versehene PDF-Dokumente können geöffnet werden, wenn Benutzer- oder Master-Kennwort bekannt sind. Der Nachteil dabei ist, dass die Verteilung von Kennwörtern problematisch werden kann, da stets ein vertraulicher Kanal erforderlich ist. Ebenso kann es passieren, dass berechtigte Dokumentempfänger Kennwörter absichtlich oder unabsichtlich mit anderen teilen.

Zertifikatsicherheit ist eine geeignete Alternative zur kennwortbasierten Sicherung. Sie basiert auf der Public-Key-Kryptografie sowie auf Zertifikaten. Ein Dokument wird für eine Reihe von Empfängern verschlüsselt, die jeweils durch ihr Zertifikat identifiziert werden. Da Zertifikate nur den öffentlichen Schlüssel *(Public Key),* aber keine vertraulichen Informationen enthalten, bedürfen sie auch bei der Verteilung keiner besonderen Sicherheit. Um ein so geschütztes Dokument zu öffnen, benötigt ein Empfänger die digitale ID mit dem privaten Schlüssel, der zum Zertifikat passt, welches zur Verschlüsselung verwendet wurde.

Zertifikatsicherheit bietet gegenüber Kennwortschutz folgende Vorteile:

- Es müssen keine Kennwörter an die Empfänger verteilt werden.

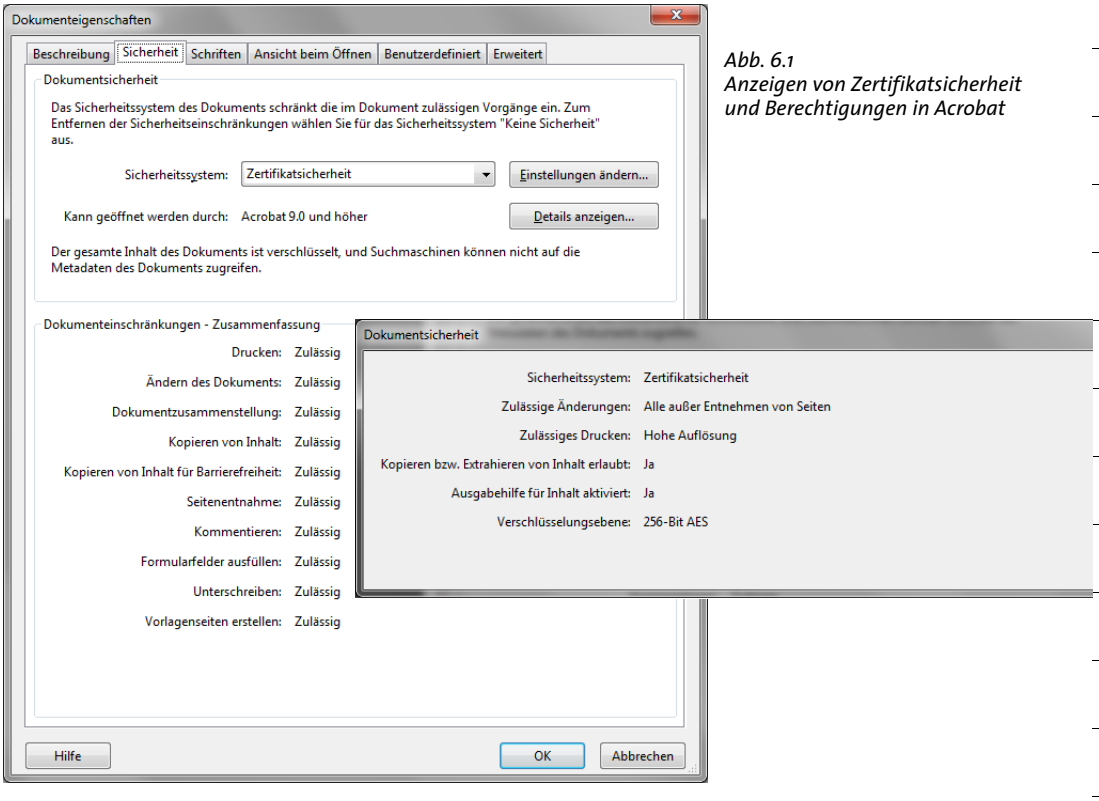


Abb. 6.1
Anzeigen von Zertifikatsicherheit und Berechtigungen in Acrobat

- Für jeden Empfänger oder Gruppe von Empfängern können individuelle Berechtigungen festgelegt werden. Berechtigungen sind nützlich, um Dokumente an Benutzer mit unterschiedlichen Benutzerrechten zu verteilen.
- Benutzer können Dokumentkennwörter nicht an unbefugte Dritte weitergeben. Software-basierte digitale IDs lassen sich zwar kopieren und weiterverteilen. Der in der ID enthaltene Name verrät jedoch den ursprünglichen Benutzer. Außerdem könnte die ID zum Fälschen seiner Signatur missbraucht werden. Schließlich können hardwarebasierte digitale IDs nicht kopiert werden.

Zertifikatsicherheit wird ab Acrobat 6 und Adobe Reader 6 unterstützt. In den folgenden Abschnitten finden Sie einen Überblick über die Zertifikatsicherheit in Acrobat. Für weitere Informationen siehe die Acrobat-Dokumentation.

Vorbereitungen für Zertifikatsicherheit. Zur Verwendung von Zertifikatsicherheit benötigen Sie digitale Zertifikate. Genauer gesagt benötigen Sie eine digitale ID (mit einem öffentlichen/privaten Schlüsselpaar) für sich selbst sowie Zertifikate (nur mit einem öffentlichen Schlüssel) für jeden Empfänger. Zertifikate können auf zwei Arten erzeugt werden:

- Selbst signiert, z.B. in Acrobat erzeugt: erhalten Sie Zertifikate direkt von den Empfängern, ist dieses Verfahren einfach und mit keinerlei Zusatzkosten verbunden. Wenn eine digitale ID jedoch verloren geht, kann sie nicht wiederhergestellt werden; verschlüsselte Dokumente lassen sich dann nicht mehr öffnen.
- Digitale IDs einer kommerziellen CA sind kostenpflichtig erhältlich, können aber bei Verlust ersetzt werden. Werden AATL-Zertifikate eingesetzt, können diese auch zum digitalen Signieren von Dokumenten verwendet werden, so dass die Validierung in Acrobat keine zusätzliche Konfiguration erfordert (siehe Abschnitt 7.1.3, »Vertrauenswürdige Stammzertifikate in Acrobat (Trusted Root Certificates)«, Seite 93).

Zum Erstellen geschützter Dokumente benötigen Sie lediglich die Zertifikate mit dem öffentlichen Schlüssel der Empfänger. Dies unterscheidet sich von den Anforderungen zum Öffnen geschützter Dokumente, wo jeder Empfänger einschließlich Ihnen selbst die entsprechende digitale ID mit dem privaten Schlüssel benötigt. Da Zertifikate keine vertraulichen Informationen enthalten, erfordern sie kein Kennwort und können frei verteilt werden, während digitale IDs in der Regel mit Kennwort oder PIN geschützt sind.

Konfigurieren der eigenen digitalen ID für die Entschlüsselung. Acrobat unterstützt verschiedene Methoden zum Konfigurieren einer digitalen ID, um damit geschützte Dokumente öffnen zu können. Zum Erzeugen oder Installieren Ihrer eigenen digitalen ID gehen Sie folgendermaßen vor:

- Acrobat XI/DC: *Bearbeiten, Voreinstellungen..., Unterschriften, Identitäten & vertrauenswürdige Zertifikate, Mehr..., Digitale IDs, ID hinzufügen*.
- Acrobat X: *Bearbeiten, Schutz, Sicherheitseinstellungen, Digitale ID, ID hinzufügen*.

Im sich darauf öffnenden Dialogfenster *Digitale ID hinzufügen* können Sie entweder eine vorhandene ID aus einer Datei hinzufügen oder eine neue selbst signierte ID erzeugen.

Um ein vertrauenswürdiges Zertifikat zu exportieren oder ein Zertifikat für Ihre eigene digitale ID zu erzeugen (damit andere für Sie Dokumente verschlüsseln können), gehen Sie folgendermaßen vor:

- Acrobat XI/DC: *Bearbeiten, Voreinstellungen..., Unterschriften, Identitäten & vertrauenswürdige Zertifikate, Weitere..., Vertrauenswürdige Zertifikate* (für die Zertifikate anderer Personen) oder *Digitale IDs* (für Ihr eigenes Zertifikat), wählen Sie die ID oder das Zertifikat und dann *Zertifikatdetails*.
- Acrobat X: *Bearbeiten, Schutz, vertrauenswürdige Identitäten verwalten..., Anzeigen: Zertifikate,* wählen Sie ein Zertifikat, *Zertifikat anzeigen..., Exportieren...* .

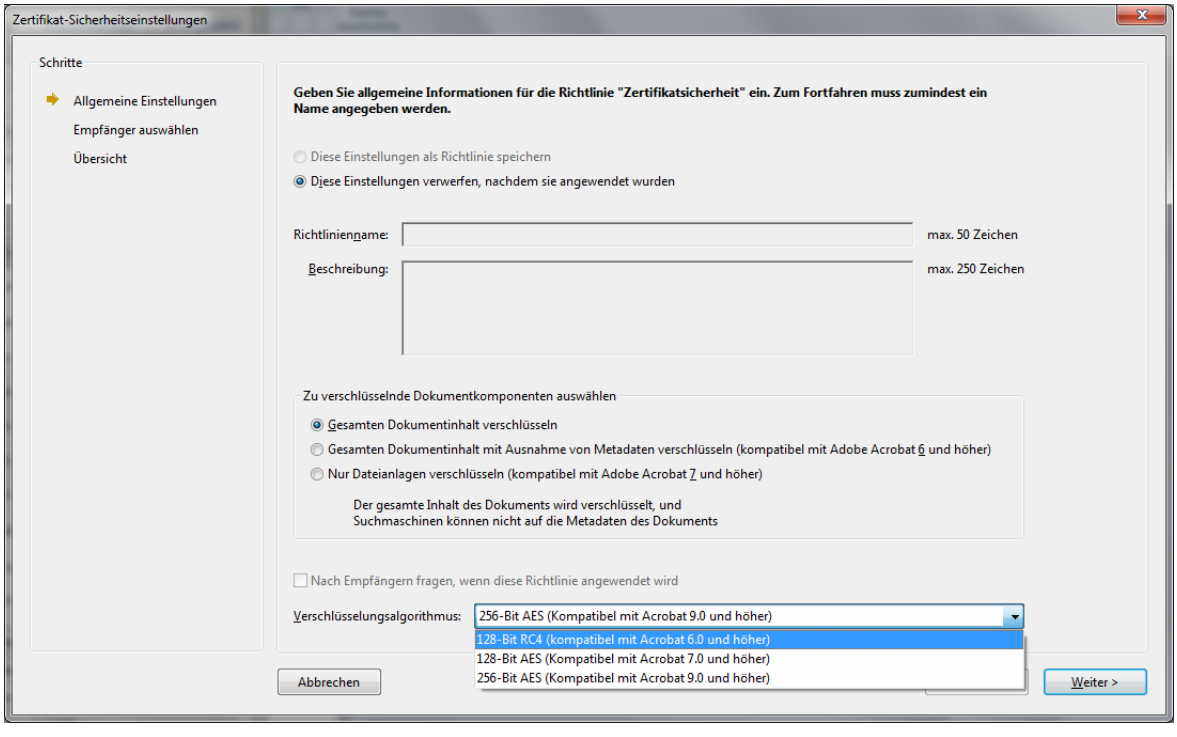
Dies öffnet die Zertifikatanzeige, in der Sie auf *Exportieren...* klicken und dann auf *Exportierte Daten in Datei speichern: Zertifikatdatei* (nicht *Certificate Message Syntax (CMS) - PKCS#7)*. Das exportierte Zertifikat kann als Empfängerzertifikat verwendet werden, sofern es noch nicht abgelaufen ist.

Hinweis *Acrobat kann auch digitale IDs im Zertifikatspeicher von Windows nutzen.*

Anwenden von Zertifikatsicherheit in Acrobat. Sobald Sie Ihre eigene digitale ID konfiguriert und die Empfängerzertifikate erhalten haben, können Sie ein PDF-Dokument mit den folgenden Schritten in Acrobat XI/DC verschlüsseln:

- Klicken Sie auf *Datei, Eigenschaften...*, wählen Sie dann in der Registerkarte *Sicherheit* unter *Sicherheitssystem* die Option *Zertifikatsicherheit*. Alternativ können Sie in Acrobat DC folgendermaßen vorgehen: *Werkzeuge, Schutz,* wählen Sie aus dem Kontextmenü *Verschlüsseln, Verschlüsseln mit Zertifikat*. Geben Sie dann den Typ der PDF-Verschlüsselung an, der für die Zertifikatsicherheit verwendet wird (siehe Abbildung 6.2).

Abb. 6.2
Zertifikatsicherheit in Acrobat mit Auswahl des PDF-Verschlüsselungsalgorithmus unten



- Wählen Sie im nächsten Dialog Ihre eigene digitale ID, um damit später das verschlüsselte Dokument wieder öffnen zu können.
- Wählen Sie dann eine beliebige Anzahl von Empfängerzertifikaten und passen Sie gegebenenfalls die Berechtigungen an. Die Empfängerzertifikate können aus dem Zertifikatspeicher von Windows importiert, aus einer Datei gelesen oder aus einem Online-Repository geholt werden.
- Beim nächsten Speichern der Datei wird diese gemäß den gewählten Sicherheitseinstellungen verschlüsselt.

Um das verschlüsselte Dokument erneut zu öffnen, benötigen Sie die digitale ID mit dem privaten Schlüssel, der zu einem der Zertifikate des Empfängers passt. Die ID kann im Zertifikatspeicher von Acrobat oder von Windows installiert sein.

Probleme beim Entschlüsseln. Wenn Acrobat ein Dokument nicht entschlüsseln kann, wird folgende Fehlermeldung ausgegeben:

Zur Verschlüsselung dieses Dokuments wurde eine digitale ID verwendet. Zur Entschlüsselung steht keine digitale ID zur Verfügung. Stellen Sie sicher, dass Ihre digitale ID korrekt installiert ist, oder wenden Sie sich an den Verfasser des Dokuments.

Jedoch erscheint diese Nachricht nicht nur, wenn die entsprechende digitale ID fehlt, sondern auch, wenn das Dokument aus anderen Gründen nicht entschlüsselt werden kann.

Inkompatibilitäten von Acrobat beim Einsatz von ECC-Empfängerzertifikaten. Acrobat XI und höher unterstützen *Elliptic Curve Cryptography (ECC)* mit den Kurven P-256-/P-384-/P-521 und anderen vom NIST empfohlenen elliptischen Kurven für die digitale Signatur und Verschlüsselung. Acrobat erzeugt jedoch ein CMS-Verschlüsselungsobjekt, das nicht konform ist zu RFC 5652 und den Ergänzungen in RFC 5753 *»Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)«*. Dies macht verschlüsselte Dokumente inkompatibel zu Drittanbieter-Software. Wir haben die Abweichungen von RFC und die Inkompatibilität in Bezug auf die Verschlüsselung mit Elliptischen Kurven an Adobe gemeldet, aber leider bleibt das Problem weiterhin in Acrobat bestehen (getestet bis Acrobat DC Classic 2015.006.30280 und Acrobat DC Continuous 15.023.20053, beide veröffentlicht im Januar 2017). Bevor das Problem in Acrobat nicht behoben ist, sollten Sie daher keine Zertifikatsicherheit mit EC-Empfängerzertifikaten verwenden.

6.2 Zertifikatsicherheit in PDF

6.2.1 CMS Enveloped Data

PDF-Zertifikatsicherheit basiert auf der *Cryptographic Message Syntax* (CMS) gemäß RFC 5652. CMS beschreibt ein Paketformat für verschiedene Verschlüsselungsfunktionen einschließlich digitaler Signaturen, Nachrichten-Authentifizierung und -Verschlüsselung. PDF-Zertifikatsicherheit verwendet die in CMS definierte Funktionalität *Enveloped Data*; verschlüsselte E-Mails funktionieren auf ähnliche Weise. Um Speicherplatz und Laufzeit zu optimieren, wird Zertifikatsicherheit auf hybride Art implementiert: zunächst wird ein Chiffrierschlüssel zufällig erzeugt und für jedes Empfängerzertifikat verschlüsselt. Dabei kommt Public-Key-Verschlüsselung mit den Algorithmen RSA oder Elliptic Curve Cryptography (ECC) zum Einsatz. Die verschlüsselten Versionen des Zufallsschlüssels werden im CMS-Objekt gespeichert. Mit dem Zufallsschlüssel wird die tatsächliche CMS-Nutzlast mit einem symmetrischen Algorithmus verschlüsselt und im CMS gespeichert. Aktuelle Implementierungen verwenden in der Regel den AES-Algorithmus für symmetrische Verschlüsselung. Für weitere Informationen zu diesem Verfahren und den verwendeten Algorithmen siehe Abschnitt 6.2.2, »Kryptografische Details«, Seite 77.

Jeder Empfänger verwendet zur Entschlüsselung des symmetrischen Schlüssels seinen privaten Schlüssel, um dann mit dem resultierenden Schlüssel die CMS-Nutzdaten zu entschlüsseln. Die Kombination von verschlüsselten Nutzdaten und einem Chiffrierschlüssel für den verschlüsselten »Content« für jeden Empfänger wird digitaler Umschlag genannt.

Da der erste Schritt bei der Verschlüsselung asymmetrisch ist und der Zufallsschlüssel nur vorübergehend gespeichert wird, kann der Verfasser eines verschlüsselten Dokuments dieses später nur dann entschlüsseln, wenn sein eigenes Zertifikat in der Liste der Empfänger enthalten ist.

Es gibt zwei Gründe für den hybriden Ansatz mit asymmetrischer und symmetrischer Verschlüsselung. Zum einen ist asymmetrische Verschlüsselung sehr langsam und nur für kleine Datenmengen geeignet. Sie wird deshalb nur für den kurzen symmetrischen Chiffrierschlüssel und nicht für die gesamten Nutzdaten verwendet. Zum anderen müssen die Nutzdaten bei dieser Vorgehensweise nur einmal symmetrisch verschlüsselt werden, während nur der kurze Chiffrierschlüssel für jeden Empfänger individuell verschlüsselt werden muss. Die Nutzdaten für jeden Empfänger zu verschlüsseln würde die Ausgabedatei erheblich vergrößern.

Die Struktur *EnvelopedData* in einem CMS-Objekt enthält eine oder mehrere Strukturen vom Typ *RecipientInfo* (siehe Abbildung 6.4). Jede enthält Informationen über ein Empfängerzertifikat – in der Regel das Zertifikat des Ausstellers (CA) und die Seriennummer – sowie einen chiffrierten Schlüssel für den Empfänger.

Vertraulichkeit der Empfängerdaten. Die Namen der Empfänger sind im CMS-Objekt nicht vorhanden. Allerdings kann der Name des Zertifikat-Ausstellers (CA) sowie die Seriennummer des Zertifikats ohne Entschlüsselung dem CMS-Objekt entnommen werden. Für Zertifikate aus einer Public Key Infrastructure (PKI) sind in diesem Fall nur der Name der CA und die Seriennummer sichtbar. Je nach PKI können diese Informationen eventuell ausreichen, um den Empfänger zu identifizieren. Bei selbst signierten Zertifikaten signiert der Zertifikatinhaber den öffentlichen Schlüssel allerdings selbst. Des-

halb sind die Namen aller Empfänger mit selbst signierten Zertifikaten im CMS-Objekt sichtbar. Manchmal kann die Sichtbarmachung von Empfängerdaten unerwünscht sein. In diesen Fällen sollten Sie in selbst signierten Zertifikaten ein Pseudonym verwenden oder diese ganz vermeiden.

Anzahl von Empfängern und Größe des CMS. Ein Dokument kann für eine beliebige Anzahl von Empfängerzertifikaten verschlüsselt werden. Da jedoch für jeden zusätzlichen Empfänger ein eindeutig chiffrierter Schlüssel eingebettet wird, erhöht sich die Länge des CMS mit der Anzahl der Empfänger. Wie stark sich die Datei vergrößert, hängt von der Länge des öffentlichen Schlüssels des Empfängers und der Menge an Informationen im Zertifikat ab. Im Allgemeinen erhöht sich die Größe der Ausgabedatei um 1-2 KB pro Empfänger.

Anwenden von CMS auf PDF-Dokumente. Ein mit Zertifikatsicherheit verschlüsseltes PDF-Dokument enthält ein oder mehrere CMS-Objekte im Eintrag *Recipients* des *Encrypt*-Dictionarys. Allerdings wird bei PDF der CMS-Mechanismus nicht direkt auf die Dokumentinhalte angewendet, sondern eine weitere, mit dem Kennwortschutz identische Ebene von Verschlüsselung hinzugefügt. Die CMS-Nutzdaten enthalten keine PDF-Objekte, sondern Schlüsselmaterial, das verwendet wird, um den Chiffrierschlüssel für PDF-Objekte abzuleiten. Die symmetrischen Algorithmen zur Verschlüsselung von PDF-Objekten sind die gleichen, die auch für den Kennwortschutz verwendet werden (siehe Tabelle 5.1). Während Zertifikatsicherheit den Schlüssel für das Dokument aus dem chiffrierten Schlüsselmaterial im CMS ableitet, wird dieser beim Kennwortschutz aus dem geheimen Kennwort abgeleitet.

PDF-Berechtigungen. Für Dokumente mit Zertifikatsicherheit können die gleichen Berechtigungen eingestellt werden wie für kennwortgeschützte PDF-Dokumente (siehe Abbildung 6.1 und »Berechtigungseinschränkungen«, Seite 63). Zusätzlich gibt es für Zertifikatsicherheit folgende Einstellungen:

- *Drucken und Bearbeiten des Dokuments sowie seine Sicherheitseinstellungen einschränken* (siehe Abbildung 6.3, oben): Ist diese Einstellung aktiviert, können Empfänger das Dokument öffnen und lesen, bestimmte Aktionen wie das Drucken und Ändern des Dokuments werden jedoch von anderen Berechtigungseinstellungen eingeschränkt. Ist diese Einstellung nicht aktiviert, haben Empfänger den vollen Zugriff auf das Dokument und können auch die Sicherheitseinstellungen ändern. Dies ist vergleichbar mit kennwortgeschützten Dokumenten, für die der Benutzer das Master-Kennwort kennt. Daher bezeichnen wir dies als *Master-Berechtigung*.

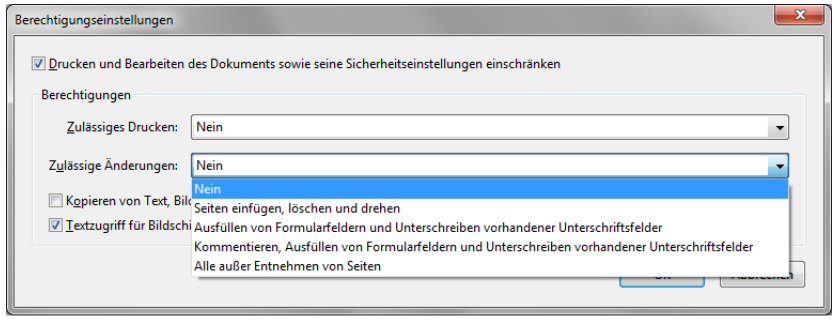


Abb. 6.3 Berechtigungseinstellungen für Zertifikatsicherheit in Acrobat

Für verschiedene Benutzer lassen sich unterschiedliche Berechtigungen festlegen. In einem Unternehmen kann zum Beispiel einem Manager die Master-Berechtigung erteilt werden, um das Dokument bearbeiten, die Verschlüsselung ändern oder eine andere Änderung vornehmen zu können, während die Kollegen nur Formularfelder ausfüllen und das Dokument signieren dürfen. Diese Flexibilität bei der Vergabe von Berechtigungen ist ein großer Vorteil der Zertifikatsicherheit gegenüber dem Kennwortschutz.

Die Berechtigungen werden in die CMS-Nutzdaten aufgenommen, die für einen bestimmten Empfänger verschlüsselt werden. Daher sind die Berechtigungen für einen bestimmten Empfänger nur nach der Entschlüsselung für diesen ersichtlich. Ohne die zugehörige digitale ID können die Berechtigungen eines Empfängers nicht abgefragt werden.

Bei der Behandlung von Berechtigungen bestehen zwischen der Zertifikatsicherheit für PDF-Dokumente und der für E-Mails signifikante Unterschiede. PDF-Dokumente können mehrere CMS-Objekte enthalten, wobei jedes Objekt wiederum eine Gruppe mehrerer Empfänger adressieren kann. Chiffrierte Schlüssel für Empfänger mit den gleichen Berechtigungen werden im selben CMS-Objekt gespeichert.

6.2.2 Kryptografische Details

Zertifikatsicherheit umfasst mehrere Verschlüsselungsschritte, die sich verschiedener Algorithmen und Schlüssellängen bedienen. Für die unten beschriebenen Schritte siehe auch Abbildung 6.4.

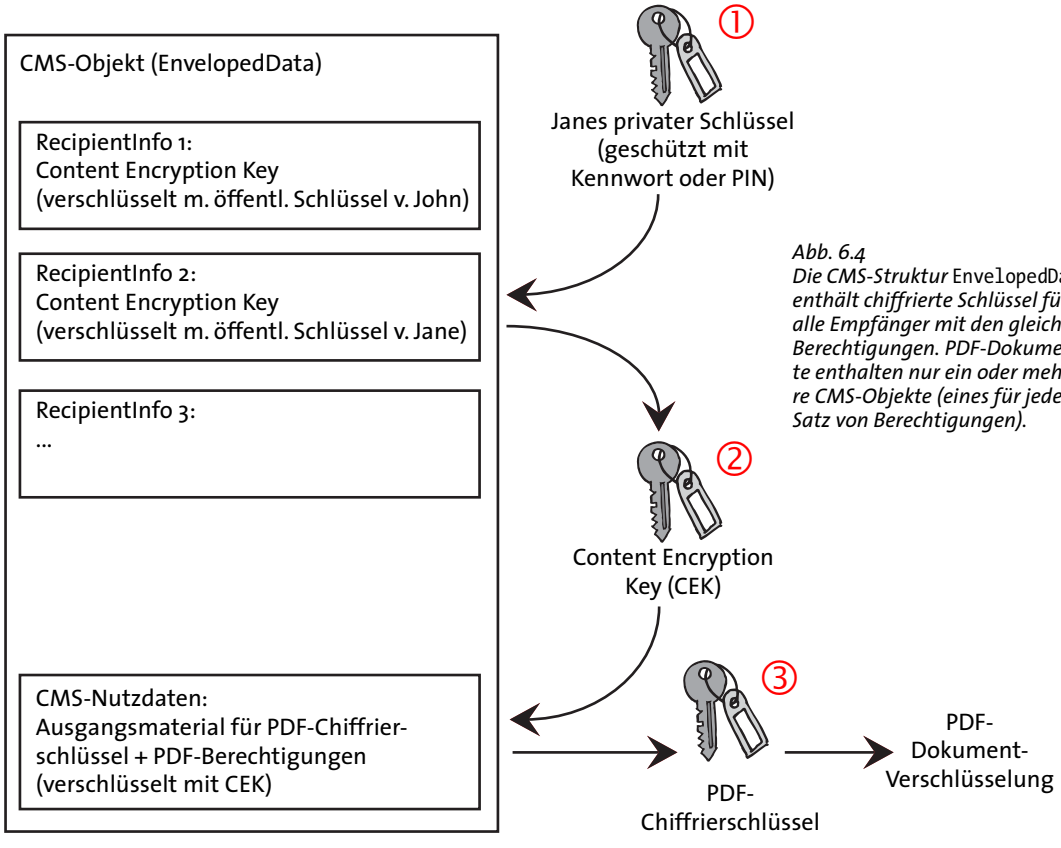


Abb. 6.4
Die CMS-Struktur `EnvelopedData` *enthält chiffrierte Schlüssel für alle Empfänger mit den gleichen Berechtigungen. PDF-Dokumente enthalten nur ein oder mehrere CMS-Objekte (eines für jeden Satz von Berechtigungen).*

Schritt 1: Public-Key-Verschlüsselung für CMS und Key-Wrap. Bei Public-Key-Verschlüsselung wird ein zufällig erzeugter Content-Chiffrierschlüssel *(Content Encryption Key, CEK)* erzeugt, wobei sich die Einzelheiten bei RSA- und ECC-Empfängerzertifikaten unterscheiden. Verschiedene Empfänger können eine Mischung aus RSA- und ECC-Schlüsseln sowie RSA-Schlüssellängen oder ECC-Kurven verwenden.

Wenn das Zertifikat des Empfängers einen öffentlichen Schlüssel für den RSA-Algorithmus (RFC 5652) enthält, wird dieser Schlüssel verwendet, um den CEK zu verschlüsseln. Die unterstützten RSA-Schlüssellängen sind nicht in der PDF-Referenz festgelegt, sondern hängen von der Acrobat-Version ab. RSA-Verschlüsselung erfordert eine Padding-Methode. Die standardmäßig eingestellte Methode PKCS#1-v1.5 wird in allen Acrobat-Versionen unterstützt. Die neuere Methode OAEP *(Optimal Asymmetric Encryption Padding)* gemäß PKCS#1 v2 (identisch mit RFC 3447) und RFC 3560 bietet hier Sicherheitsvorteile; sie kann mit der Option *rsapadding=oaep* angefordert werden. OAEP wird in Acrobat DC und darunter nicht unterstützt, jedoch in PDF-Viewern von Drittanbietern, wie z.B. Nitro PDF.

Wenn das Zertifikat des Empfängers einen öffentlichen ECC-Schlüssel (RFC 5753) enthält, werden das Schlüsselvereinbarungsschema Elliptic Curve Diffie-Hellman (ECDH) und der öffentliche Schlüssel im Empfängerzertifikat verwendet, um einen weiteren temporären Schlüssel für den Key-Chiffrierschlüssel (KEK) abzuleiten. Ein symmetrischer Verschlüsselungsalgorithmus, der so genannte Key-Wrap-Algorithmus, wird dann zur Verschlüsselung des Content-Chiffrierschlüssels (CEK) mit dem Key-Chiffrierschlüssel (Key Encryption Key, KEK) verwendet. Acrobat XI/DC verwendet als Key-Wrap-Algorithmus AES-128 oder AES-256. Die unterstützten ECC-Kurven sind nicht in der PDF-Referenz festgelegt, sondern hängen von der Acrobat-Version ab.

Schritt 2: CMS »Content«-Verschlüsselung. Der Content-Chiffrierschlüssel wird verwendet, um das PDF-Schlüsselmaterial (nicht den eigentlichen Schlüssel selbst) mit einem symmetrischen Algorithmus zu verschlüsseln, wodurch die CMS-Nutzdaten verschlüsselt werden. Bei der PDF-Zertifikatsicherheit enthält der CMS-Content keine PDF-Dokumentdaten, sondern Verschlüsselungsmaterial, aus dem der endgültige Chiffrierschlüssel für PDF-Objekte abgeleitet wird.

Für den CMS-Content kann eine Reihe von Algorithmen für die Verschlüsselung gewählt werden. Acrobat 7-X verwenden immer Triple-DES, Acrobat XI/DC verwenden AES-128. Da die Nutzdatenverschlüsselung nur einmal benötigt wird, unabhängig von der Anzahl der Empfänger, hängt die Wahl des Algorithmus nicht von den Empfängerzertifikaten ab.

Schritt 3: PDF-Verschlüsselung. Der PDF-Chiffrierschlüssel wird auf PDF-Objekte angewendet, was die Daten zur Anzeige des Dokuments liefert. Dieser Schritt wird beim Kennwortschutz identisch durchgeführt.

Der symmetrische Algorithmus und die Schlüssellänge für die Verschlüsselung von PDF-Objekten werden in der PDF-Referenz festgelegt und entsprechen einer Teilmenge der für den Kennwortschutz verwendeten (siehe Tabelle 5.1). Zur Verschlüsselung aller Objekte im PDF-Dokument wird der selbe symmetrische Algorithmus verwendet. Lediglich der Algorithmus für Schritt 3 kann in Acrobat ausgewählt werden (siehe Abbildung 6.2, unten). Bei der mit Acrobat 7 und Acrobat 9 eingeführten starken Verschlüsselung wird der PDF-Verschlüsselungsalgorithmus AES-128 bzw. AES-256 verwendet.

Algorithmen und Schlüssellängen. Tabelle 6.1 fasst Algorithmen und Schlüssellängen für diverse PDF- und Acrobat-Versionen zusammen. Alle beteiligten Algorithmen sollten von vergleichbarer Stärke sein, da das schwächste Glied das attraktivste Ziel für Angreifer darstellt. Acrobat befolgt diese Regel allerdings nicht ganz konsequent. Acrobat verwendet für die CMS-Content-Verschlüsselung zum Beispiel den Algorithmus AES-128 auch in der Kombination mit PDF-Verschlüsselung AES-256, wodurch die PDF-Verschlüsselung durch die CMS-Verschlüsselung geschwächt wird.

Tabelle 6.1 PDF- und CMS-Verschlüsselungsalgorithmen für Zertifikatsicherheit und Unterstützung in Acrobat

PDF-/Acrobat-Version und Nummer des pCOS-Algorithmus	***Schritt 1: Public-Key-Algorithmus für CMS***	***Schritt 2: CMS »Content«-Verschlüsselung***	***Schritt 3: PDF-Verschlüsselung***
PDF 1.4 (Acrobat 5), pCOS-Algorithmus 5	*Acrobat 6 und höher: 2048-Bit RSA*	*Acrobat 6-X: Triple-DES* *Acrobat XI/DC: 128-Bit AES[1]* *PLOP: nur Lesen wird unterstützt*	*128-Bit RC4 (schwach; in PDF 2.0 als veraltet deklariert)*
PDF 1.6 (Acrobat 7), pCOS-Algorithmus 6	*Acrobat 7 und höher: bis 4096-Bit RSA* *Acrobat 8 und höher: bis 8192-Bit RSA[2]*	*Acrobat 7-X: Triple-DES* *Acrobat XI/DC: 128-Bit AES* *PLOP: 128-Bit AES*	*128-Bit AES (in PDF 2.0 als veraltet deklariert)*
PDF 1.7ext3 (Acrobat 9), pCOS-Algorithmus 10	*Acrobat 9 und höher: bis 8192-Bit RSA[3],[4]* *Acrobat XI/DC[5]: ECC mit den Kurven P-256/P-384/P-521[6],[7]*	*Acrobat XI/DC: 128-Bit AES* *PLOP: 256-Bit AES*	*256-Bit AES (stark)*

1. Die Verwendung von AES-128 für Algorithmus 5 bedeutet, dass mit der Einstellung »128-Bit RC4 (kompatibel mit Acrobat 6 und höher)« in Acrobat XI/DC verschlüsselte Dokumente sich dennoch nicht in Acrobat 6 öffnen lassen, da diese Version AES nicht unterstützt.
2. RSA-8192-Schlüssel erfordern Acrobat X oder höher und werden von Acrobat unter OS X nicht unterstützt.
3. Bei der Entschlüsselung von Dokumenten mit einer ID im Windows-Zertifikatspeicher unterstützt Acrobat nur Schlüssel, bei denen die Länge ein Vielfaches von 8 Bit beträgt.
4. RSA mit OAEP-Padding wird in Acrobat DC und darunter nicht unterstützt.
5. Acrobat 9/X unterstützen keine ECC-Verschlüsselung. Wenn jedoch ein Dokument sowohl für ein RSA- als auch ein ECC-Empfängerzertifikat verschlüsselt wurde und beide Empfänger unterschiedliche Berechtigungen haben, kann das Dokument in Acrobat 9/X mit einer RSA-ID geöffnet werden.
6. Acrobat XI/DC erzeugen inkompatible Ausgabe für ECC-Verschlüsselung, siehe »Inkompatibilitäten von Acrobat beim Einsatz von ECC-Empfängerzertifikaten«, Seite 74.
7. Die Ableitungsfunktion `dhSinglePass-stdDH-sha512kdf` *für den Schlüssel (optional in RFC 5753) wird in MSCAPI nicht unterstützt. Deshalb können solche Dokumente von PLOP nicht mit* `engine=mscapi`*, sondern nur mit* `engine=builtin` *entschlüsselt werden.*

6.3 Anwendungsfälle für Zertifikatsicherheit

In diesem Abschnitt werden Anwendungsfälle vorgestellt, die von den Vorteilen der Zertifikatsicherheit profitieren. In den meisten Fällen sollten folgenden Fragen untersucht werden:

- Ist es erforderlich, das eigene Zertifikat des Verfassers in der Liste der Empfänger aufzunehmen? Wenn es nicht enthalten ist, kann der Verfasser das geschützte Dokument nicht öffnen.
- Welche Berechtigungen gelten für jeden Empfänger oder jede Gruppe von Empfängern?

Verteilen vertraulicher Dokumente an eine geschlossene Gruppe von Empfängern. Mitglieder einer Gruppe wollen vertrauliche Dokumente austauschen, so dass alle anderen Mitglieder der Gruppe die Dokumente verwenden können. Die PDF-Dateien werden für die Zertifikate aller Gruppenmitglieder verschlüsselt. Wenn der Verfasser eines Dokuments sein eigenes Zertifikat bei der Verschlüsselung der Datei hinzufügt, ist nur eine einzige Version des Dokuments erforderlich. Obwohl die Zahl der Empfänger nicht eigentlich begrenzt ist, sollte im Auge behalten werden, dass durch jeden zusätzlichen Empfänger das Dokument leicht vergrößert wird.

In einem ähnlichen Anwendungsfall dürfen einige Empfänger (die Manager) das Dokument ändern, während normalen Mitarbeitern nur das Ausfüllen von Formularen und das Signieren von PDF-Dokumenten gestattet ist. Diese Unterscheidung kann mit separaten Empfängergruppen erreicht werden, wobei jeder Gruppe die entsprechenden Berechtigungen zugewiesen werden.

Wenn die Anzahl der Gruppenmitglieder groß wird (Tausende von Empfängern), kann die Gruppe in kleinere Gruppen aufgeteilt werden. Eine kleine Anzahl von Empfängern in jeder Teilgruppe minimiert die Dateigröße, während eine große Anzahl von Empfängern in jeder Teilgruppe die Anzahl der verschiedenen geschützten Versionen eines Dokuments reduziert.

Serielles Signieren eines vertraulichen Dokuments. Ein vertrauliches Dokument wird für mehrere Empfänger verschlüsselt. Die Empfänger sollen das Dokument digital signieren, weitere Änderungen sind jedoch nicht erlaubt. Dazu werden die Berechtigungen entsprechend gesetzt. Empfänger können die gleiche digitale ID zum Verschlüsseln und Signieren des Dokuments verwenden, sofern beide Berechtigungen im Zertifikat gewährt werden.

Digital Rights Management (DRM). Dokumente mit kommerziellen Inhalten werden an zahlende Kunden verteilt. Jeder Abonnent oder Käufer erhält hierbei ein geschütztes PDF, das für sein persönliches Zertifikat verschlüsselt wurde. Um eine individuelle Dokumentversion für jeden Empfänger zu erstellen, können viele geschützte Versionen des selben Dokuments erzeugt werden. Um die Kunden an der Manipulation des Dokuments zu hindern, kann die Berechtigungsbeschränkung *nomaster* gesetzt werden.

Sicheres Speichern und Archivieren. In diesem Szenario erhält ein Archiv Dokumente, die geschützt werden müssen. Jedes archivierte Dokument wird für das Zertifikat des Archivbesitzers verschlüsselt. Für das archivierte Dokument muss nur eine einzige geschützte Version erstellt werden.

Verteilung von Rechnungen und Anschreiben. Eine kundenspezifische Rechnung, ein Anschreiben oder ein Transaktionsdokument wird für das Zertifikat des Kunden verschlüsselt, um die Vertraulichkeit zu gewährleisten. Eine einzige geschützte Version jedes Dokuments wird erstellt und an den Kunden geschickt. Um die Kunden an der Manipulation des Dokuments zu hindern, kann die Berechtigungsbeschränkung *nomaster* gesetzt werden.

6.4 Zertifikatsicherheit mit PLOP

PDF-Verschlüsselungsalgorithmus und Schlüssellänge. PLOP wendet Zertifikatverschlüsselung immer in Kombination mit den starken Algorithmen AES-128 oder AES-256 an, jedoch nie mit dem schwachen RC4-Algorithmus. Der PDF-Verschlüsselungsalgorithmus lässt sich mit der Option *encryption* von *PLOP_create_document( )* auswählen:

- Bei *encryption=algo6:* die PDF-Version wird auf PDF 1.6 erhöht, sofern erforderlich, und Zertifikatverschlüsselung mit AES-128 gemäß Acrobat 7 (pCOS-Algorithmus 6) wird angewendet.
- Bei *encryption=algo10:* (Standardwert): die PDF-Version wird auf PDF 1.7ext3 erhöht, sofern erforderlich, und Zertifikatverschlüsselung mit AES-256 gemäß Acrobat 9 (pCOS-Algorithmus 10) wird angewendet.

Der PDF-Verschlüsselungsalgorithmus (d.h. AES-128 oder AES-256) wird auch zur CMS-Content-Verschlüsselung verwendet; schwache Algorithmen werden nicht eingesetzt.

Angabe der Empfängerzertifikate. Für jeden Empfänger muss ein Zertifikat bereitgestellt werden, das den öffentlichen Schlüssel des Empfängers enthält. Im Gegensatz zu einer digitalen ID enthält ein Zertifikat keinen privaten Schlüssel und muss deshalb nicht geschützt werden. Das PDF-Dokument wird so verschlüsselt, dass es nur von den angegebenen Empfängern mit dem privaten Schlüssel entschlüsselt werden kann, der zum öffentlichen Schlüssel in ihrem Zertifikat gehört.

Bevor ein Ausgabedokument erstellt werden kann, muss jeder Empfänger mit einem Aufruf von *PLOP_add_recipient( )* angegeben werden (der vollständige Beispielcode ist im Minibeispiel *certsec* verfügbar, das in allen PLOP-Paketen enthalten ist):

```
if (plop.add_recipient("certificate={filename=demo_recipient_1.pem}") == -1)
{
        /* Warnung ausgeben und fortfahren */
        System.err.print("Warning: ", plop.get_errmsg());
}
...
if (plop.create_document(out_filename, optlist) == -1) {
        System.err.println("Error: " + plop.get_errmsg());
        plop.delete();
        System.exit(2);
}
```

Die Angabe mindestens eines Empfängers aktiviert die Zertifikatsicherheit. Sobald eine Liste der Empfänger erstellt wurde, wird sie auf alle nachfolgend erzeugten Dokumente angewendet, bis eine neue Liste mit zusätzlichen Anrufen von *PLOP_add_recipient( )* erstellt wird.

Beim PLOP-Kommandozeilen-Tool können Empfänger mit der Option *--recipient* angegeben werden.

Anforderungen an Empfängerzertifikate. Ein zum Schutz von PDF-Dokumenten mit PLOP verwendetes Empfängerzertifikat muss folgende Anforderungen erfüllen:

- Wenn das Zertifikat eine Erweiterung der Schlüsselverwendung enthält, muss diese das Zertifikat für die Verschlüsselung aktivieren. Zertifikate, die nur digitale Signaturen erlauben, können nicht für die Verschlüsselung verwendet werden.

- Das Zertifikat muss gültig sein, d.h. sein Verfallsdatum darf noch nicht erreicht worden sein.
- RSA-Schlüssel werden standardmäßig nur dann akzeptiert, wenn die Schlüssellänge ein Vielfaches von 8 Bit beträgt. Schlüssel mit Sonderlängen werden mit der Option *conformance=extended* akzeptiert. Allerdings können die resultierenden Dokumente in Acrobat nicht mit einer digitalen ID im Zertifikatspeicher von Windows geöffnet werden, sondern nur mit einer ID im Zertifikatspeicher von Acrobat. Es wird empfohlen, Schlüssel zu verwenden, bei denen die Länge ein Vielfaches von 64 Bit beträgt.
- ECC-Empfängerzertifikate mit anderen Kurven als P-256/P-384/P-521 werden standardmäßig abgelehnt, aber mit der Option *conformance=extended* akzeptiert. Die verschlüsselten Dokumente können jedoch nicht mit Acrobat XI/DC geöffnet werden.

Zugriffsberechtigungen. Berechtigungsbeschränkungen, wie Drucken nicht zulässig, können in der Option *permissions* von *PLOP_add_recipient()* für jeden Empfänger separat angegeben werden. Sofern nicht anders angegeben, sind standardmäßig alle Aktionen erlaubt. Tabelle 5.3 zeigt die unterstützten Schlüsselwörter zum Setzen von Berechtigungen.

Beachten Sie, dass in Acrobat die vier Berechtigungseinschränkungen für Dokumentänderungen nicht alle separat gesetzt werden können, sondern einige zu Gruppen zusammengefasst sind. Tabelle 5.4 listet Acrobat-Einstellungen (die Werte von »Änderungen erlaubt« in Abbildung 6.3) mit den entsprechenden Kombinationen von PLOP-Schlüsselwörtern für Zugriffsberechtigungen auf.

Beispieloptionsliste für *PLOP_add_recipient()*, die nur das Ausfüllen von Formularfeldern und das Signieren erlaubt:

```
certificate={filename=demo_recipient_1.pem} permissions={nomodify noannots noassemble}
```

Kryptografische Engines. Mit den in Abschnitt 7.2, »Signieren von Dokumenten mit PLOP DS«, Seite 96 beschriebenen Engines *builtin* und *mscapi* können auch Empfängerzertifikate für die Verschlüsselung ausgewählt werden.

Die Engine kann mit der Option *engine* von *PLOP_add_recipient()* angegeben werden und bestimmt die Unteroptionen für die Auswahl der Zertifikate:

- Mit *engine=builtin* (Standardwert) müssen Zertifikate als X.509-Dateien in PEM- oder DER-Kodierung übergeben werden, z.B.

  ```
  engine=builtin certificate={filename=demo_recipient_1.pem}
  ```

  Die angegebene Zertifikatdatei muss genau ein Verschlüsselungszertifikat enthalten.
- Mit *engine=mscapi* können Zertifikate aus dem Zertifikatspeicher von Windows geholt werden. Sie werden anhand des Namens für den Zertifikatspeicher und des Subjektnamens des Empfängers im Zertifikat ausgewählt, z.B.

  ```
  engine=mscapi certificate={store=My subject={PLOP Demo Recipient 1}}
  ```

Entschlüsselung geschützter Dokumente. Um ein mit Zertifikatsicherheit geschütztes Dokument zu entschlüsseln, benötigen Sie eine digitale ID, die zu einem der Empfängerzertifikate im Dokument passt. Die Option *digitalid* von *PLOP_open_document()* muss zusammen mit dem zugehörigen Kennwort für die ID übergeben werden, z.B.

```
digitalid={filename=demo_recipient_1.p12} password=demo
```

Wird die digitale ID aus dem Zertifikatspeicher von Windows abgerufen, werden alle IDs im angegebenen Speicher zum Entschlüsseln des Dokuments durchprobiert. Die Unteroption *subject* zur Auswahl einer ID ist daher nicht erforderlich:

```
engine=mscapi digitalid={store=My}
```

Da *My* der Standardname für den Speicher ist, kann dies folgendermaßen weiter abgekürzt werden:

```
engine=mscapi
```

Erforderliche Zugangsdaten für verschiedene PLOP-Operationen. Um die Vorgaben des Verfassers zu respektieren, die in den Sicherheitseinstellungen eines PDF-Dokuments hinterlegt sind, können an einem mit Zertifikatsicherheit verschlüsselten Dokument unter Umständen nicht alle Operationen durchgeführt werden. PLOP verarbeitet Dokumente mit Zertifikatsicherheit gemäß folgender Regeln:

- Die Abfrage des Verschlüsselungsstatus mit dem pCOS-Pseudo-Objekt *encrypt/algorithm* usw. ist unabhängig von der Verfügbarkeit einer geeigneten digitalen ID immer erlaubt.
- Die Abfrage von anderen Dokumenteigenschaften mit der pCOS-Schnittstelle erfordert eine geeignete digitale ID, d.h. eine ID mit einem privaten Schlüssel, der zu einem der öffentlichen Schlüssel des Empfängers im verschlüsselten Dokument passt. Dies lässt sich mit *pcosmode=1* prüfen.
- Das Signieren eines Dokuments im inkrementellen Update-Modus erfordert ebenfalls eine geeignete digitale ID. Darüber hinaus muss die Berechtigungseinstellung *noannots* im Dokument auf *false* gesetzt sein. Dies lässt sich mit dem pCOS-Pseudo-Objekt *encrypt/noannots* prüfen.
- Die Bearbeitung des Dokuments auf andere Weise, z.B. das Entfernen von Verschlüsselung oder das Ändern von Berechtigungen, erfordert eine geeignete digitale ID. Zusätzlich muss die Master-Berechtigung für die ID gesetzt sein, mit der das Dokument geöffnet wird. Dies lässt sich mit *pcosmode=2* prüfen.

Tabelle 6.2 zeigt die jeweiligen Voraussetzungen für die verschiedenen Operationen.

Tabelle 6.2 Erforderliche digitale IDs für verschiedene Operationen an verschlüsselten Dokumenten

verfügbare ID, Master-Berechtigung und pCOS-Modus	***Abfrage des Verschlüsselungsstatus mit pCOS***	***Abfrage anderer Dokumenteigenschaften mit pCOS***	***Signieren im Update-Modus***	***andere Verarbeitung z.B. Verschlüsselung ändern***
keine (pCOS-Modus 0)	*ja*	*nein*	*nein*	*nein*
geeignete ID verfügbar und Master-Berechtigung nicht aktiviert (pCOS-Modus 1)	*ja*	*ja*	*nur bei* `noannots=false`	*nein*
geeignete ID verfügbar, Master-Berechtigung für diese ID aktiviert (pCOS-Modus 2)	*ja*	*ja*	*ja*	*ja*

Abfragen von Empfängern und Berechtigungen mit pCOS. Mit dem pCOS-Pseudo-Objekt *encrypt/recipients* können Sie das Dokument auf Zertifikatsicherheit prüfen. Ist der Wert von

```
length:encrypt/recipients
```

größer als Null, enthält jeder Eintrag in diesem Array ein CMS-Objekt für eine Gruppe von einem oder mehreren Empfängern mit identischen Berechtigungen. Da jedes CMS-Objekt einen oder mehrere Empfänger enthalten kann, gibt die Array-Länge nicht notwendigerweise die Gesamtzahl der Empfänger an.

Mit dem pCOS-Pseudo-Objekt *pcosmode* lässt sich prüfen, ob eine geeignete digitale ID zum Öffnen des Dokuments übergeben wurde und die Master-Berechtigung gesetzt ist:

- Minimaler pCOS-Modus *(pcosmode=0):* es wurde keine geeignete digitale ID übergeben.
- Eingeschränkter pCOS-Modus *(pcosmode=1):* es wurde eine geeignete digitale ID zum Öffnen des Dokuments übergeben, aber die Master-Berechtigung wurde für diesen Empfänger im Dokument nicht gesetzt. Das Signieren des Dokuments ist nur erlaubt, wenn die Berechtigung *noannots* auf *false* gesetzt ist.
- Vollständiger pCOS-Modus *(pcosmode=2):* es wurde eine geeignete digitale ID zum Öffnen des Dokuments übergeben und die Master-Berechtigung für diesen Empfänger wurde im Dokument gesetzt. Alle Dokumentberechtigungen gelten ohne Einschränkung und alle PLOP-Operationen sind erlaubt.

Zugriffsberechtigungen können mit folgenden Einträgen im pCOS-Pseudo-Objekt *encrypt* geprüft werden (z.B. *encrypt/noassemble):*

```
noaccessible, noannots, noassemble, nocopy, noforms, nohiresprint, nomodify, noprint
```

Beachten Sie, dass es kein Pseudo-Objekt *encryption/nomaster* gibt, da der Status des Flags für die Master-Berechtigung mit *pcosmode=2* geprüft werden kann. Für weitere Informationen siehe die pCOS-Pfadreferenz. Das Minibeispiel *dumper* enthält Beispielcode zur Identifizierung von Dokumenten mit Zertifikatsicherheit.

6.5 Anwenden von Zertifikatsicherheit auf der Kommandozeile

In den Beispielen für Kommandozeilenaufrufe unten werden die langen Optionsnamen verwendet; für die Kurzform der Optionen siehe Abschnitt 3.1, »PLOP- und PLOP DS-Kommandozeilen-Optionen«, Seite 39.

Verschlüsselung. Empfängerzertifikate können mit der Kommandozeilenoption *--recipient* oder der Kurzform *-r* angegeben werden. Diese Option kann für mehrere Empfänger wiederholt werden.

Verschlüsseln eines Dokuments für einen einzelnen Empfänger, sofern das Zertifikat in Dateiform vorliegt:

```
plop --recipient "certificate={filename=demo_recipient_1.pem}" ↵
        --outfile encrypted.pdf input.pdf
```

Verschlüsseln eines Dokuments für einen Empfänger und Einschränken seiner Berechtigungen, so dass Drucken und Kopieren nicht erlaubt sind:

```
plop --recipient "certificate={filename=demo_recipient_1.pem permissions={noprint
        nocopy}}" --outfile encrypted.pdf input.pdf
```

Verschlüsseln eines Dokuments für zwei Empfänger, sofern die Zertifikate in Dateiform vorliegen:

```
plop --recipient "certificate={filename=demo_recipient_1.pem}" ↵
     --recipient "certificate={filename=demo_recipient_2.pem}" ↵
     --outfile encrypted.pdf input.pdf
```

Verschlüsseln eines Dokuments für eine große Anzahl von Empfängern: in diesem Fall sind Response-Dateien für das PLOP-Kommandozeilen-Tool nützlich (siehe »Response-Dateien«, Seite 42). Erzeugen einer Textdatei *recipients.txt* mit allen erforderlichen Empfängeroptionen:

```
--recipient "certificate={filename=demo_recipient_1.pem}"
--recipient "certificate={filename=demo_recipient_2.pem}"
--recipient "certificate={filename=demo_recipient_3.pem}"
--recipient "certificate={filename=demo_recipient_4.pem}"
...
```

Als nächstes geben Sie den Namen dieser Response-Datei im PLOP-Kommandozeilenaufruf mit einem vorangestellten @-Zeichen ein:

```
plop @recipients.txt --outfile encrypted.pdf input.pdf
```

Verschlüsseln eines Dokuments für einen einzelnen Empfänger, für den ein Zertifikat im Zertifikatspeicher von Windows vorliegt:

```
plop --recipient "engine=mscapi certificate={store=My subject={PLOP Demo Recipient 1}}" ↵
     --outfile encrypted.pdf input.pdf
```

Verschlüsseln und Signieren eines Dokuments (für die Signaturoptionen siehe Abschnitt 7.2.2, »Signieren mit der integrierten Engine«, Seite 97). Dazu benötigt man das Empfängerzertifikat sowie die digitale ID des Unterzeichners:

```
plop --recipient "certificate={filename=demo_recipient_1.pem}" ↵
     --signopt "update=false digitalid={filename=demo_signer_rsa_2048.p12} password=demo" ↵
     --outfile signed+encrypted.pdf input.pdf
```

Zugriffsberechtigungen. Verschlüsseln eines Dokuments für zwei Empfänger, wobei dem ersten der volle Zugriff gewährt wird und dem zweiten Empfänger nur erlaubt wird, das Dokument zu signieren. Da das Berechtigungsschlüsselwort *noforms* fehlt, ist das Ausfüllen von Formularen und das Signieren für den zweiten Empfänger erlaubt:

```
plop --recipient "certificate={filename=demo_recipient_1.pem}" ↵
     --recipient "certificate={filename=demo_recipient_2.pem} ↵
        permissions={nomodify nocopy noannots noassemble}" ↵
     --outfile encrypted.pdf input.pdf
```

Verschlüsseln eines Dokuments für einen einzelnen Empfänger, der sich das Dokument nur anzeigen darf:

```
plop --recipient "certificate={filename=demo_recipient_1.pem}" ↵
     permissions={noprint nomodify nocopy noannots noassemble noforms}" ↵
     --outfile encrypted.pdf input.pdf
```

Entschlüsselung. Entschlüsseln eines Dokuments, das mit Zertifikatsicherheit geschützt ist, und Erstellen einer ungeschützten Version, sofern eine geeignete digitale ID in Dateiform vorliegt:

```
plop --inputopt "digitalid={filename=demo_recipient_1.p12} password=demo" ↵
     --outfile decrypted.pdf encrypted.pdf
```

Entschlüsseln eines geschützten Dokuments und Erstellen einer ungeschützten Version, sofern eine geeignete digitale ID im Zertifikatspeicher *My* von Windows vorliegt. Da die Option *store* standardmäßig auf *My* gesetzt ist und PLOP automatisch eine geeignete ID findet, kann die Option *digitalid* weggelassen werden. Die Option *password* kann auch weggelassen werden, da der private Schlüssel durch die Windows-Anmeldedaten geschützt ist:

```
plop --inputopt "engine=mscapi" --outfile decrypted.pdf encrypted.pdf
```

7 Digitale Signaturen mit PLOP DS

Hinweis *Das digitale Signieren von PDF-Dokumenten wird nur von PDFlib PLOP DS, nicht aber vom Basisprodukt PLOP unterstützt.*

7.1 Einführung

7.1.1 Grundbegriffe der digitalen Signatur

Ausführliche Informationen zu digitalen Signaturen würden den Rahmen dieses Handbuchs sprengen. Jedoch erläutern wir in diesem Kapitel die wesentlichen Komponenten, die beim digitalen Signieren von PDF-Dokumenten mit PLOP DS eine Rolle spielen. Zusammen bilden diese Komponenten die *Public Key Infrastructure* (PKI).

Digitale Signaturen basieren auf *Public Key Cryptography,* auch asymmetrische Verschlüsselung genannt. Sie arbeitet mit einem privaten Schlüssel *(private key),* der nur der Person bekannt ist, die ein Dokument unterschreibt (signiert), und einem öffentlichen Schlüssel *(public key),* der für jeden zugänglich ist, um die Signaturen zu validieren.

Zertifikate. Öffentliche Schlüssel werden in der Regel in einem Zertifikat verteilt, das den öffentlichen Schlüssel sowie Namen und Kontaktinformationen des Unterzeichners enthält. Um gefälschte Zertifikate zu vermeiden, wird dieses Informationspaket wiederum von einem vertrauenswürdigen, unbeteiligten Dritten signiert, der ein Zertifikat für eine Person oder eine andere Instanz, z.B. ein Unternehmen oder einen Server, ausstellt. Solche vertrauenswürdigen Anbieter nennt man Zertifizierungsstellen *(Certificate Authority, CA)* oder *Trust Center (TC).* Das eigene Zertifikat der CA ist das Stammzertifikat, das in der Regel auf der Website der CA zum allgemeinen Download zur freien Verfügung steht. Zertifikate werden normalerweise im Format X.509 gespeichert.

Da zur Verschlüsselung eines Dokuments mit Zertifikatsicherheit nur der öffentlichen Schlüssel erforderlich ist, genügt das Empfängerzertifikat. Auf der anderen Seite erfordert die Entschlüsselung eines solchen Dokuments den privaten Schlüssel, der nur in der digitalen ID des Empfängers verfügbar ist (siehe unten).

Zertifikatskette. Ein von einer CA ausgestelltes Signaturzertifikat gilt als vertrauenswürdig, wenn die ausstellende CA oder die übergeordnete CA, die das Zertifikat der zwischengeschalteten CA ausgestellt hat, als vertrauenswürdig gilt. Die Liste der Zertifikate, die durch das Signieren des jeweils nächsten Zertifikats verbunden sind, vom Zertifikat der Stamm-CA bis hinunter zum tatsächlich für ein Dokument verwendetes Endbenutzer-Zertifikat, wird als Zertifikatskette bezeichnet. Das höchste CA-Zertifikat in der Kette ist das Stammzertifikat. Damit eine Signatur als gültig eingestuft wird, müssen alle Zertifikate in der Kette gültig sein.

Digitale IDs. Zum Verständnis des Konzepts ist es wichtig, zwischen Zertifikaten und dem Paket aus Zertifikat und zugehörigem privaten Schlüssel, der sogenannten digitalen ID, zu unterscheiden. Während Zertifikate frei verteilt werden können, müssen digitale IDs sorgfältig geschützt werden, da sie vertrauliche Informationen (nämlich den privaten Schlüssel) enthalten. Um auf den privaten Schlüssel in einer digitalen ID zugreifen zu können oder um ein Dokument mit Zertifikatsicherheit zu entschlüsseln,

wird ein Kennwort oder eine Passphrase benötigt. Ein weit verbreitetes Speicherformat für digitale IDs ist PKCS#12 (unter Windows auch PFX genannt). Beachten Sie, dass zwischen Zertifikaten und digitalen IDs nicht immer klar unterschieden wird: oft spricht man von *ein Dokument mit einem Zertifikat signieren,* wenn es richtigerweise *mit einer digitalen ID signieren* heißen müsste.

Prüfung der Zertifikatsperrung (Certificate Revocation). Zertifikate sind für einen bestimmten Zeitraum gültig. Sobald ihre Gültigkeitsdauer überschritten ist oder sie explizit von der Zertifizierungsstelle gesperrt werden, werden sie ungültig. Ein Zertifikat zu sperren kann erforderlich sein, wenn der Zertifikatinhaber die zugehörige Organisation verlassen hat oder der private Schlüssel kompromittiert wurde.

Die Überprüfung von Zertifikaten erfolgt in der Regel mit einer Online-Abfrage über ein sogenanntes OCSP-Protokoll *(Online Certificate Status Protocol)* oder über Zertifikatsperrlisten *(Certificate Revocation Lists, CRLs)*. Für weitere Informationen hierzu siehe »Überblick über OCSP«, Seite 117 und »Überblick über Zertifikatsperrlisten (CRLs)«, Seite 120.

Zeitstempel. Zeitstempel wenden eine digitale Signatur auf den Inhalt einer Datei zu einem bestimmten Zeitpunkt an, wobei die Zeit von einer vertrauenswürdigen und akkuraten Zeitquelle bezogen werden kann. Zeitstempel lassen sich in eine normale Signatur integrieren, um zu bescheinigen, dass die Signatur und das signierte Dokument vor einem bestimmten Zeitpunkt vorhanden war. Sie können auch separat auf ein Dokument angewendet werden. Für weitere Informationen zu Zeitstempel-Servern und -protokollen siehe Abschnitt 7.5.1, »Konfiguration von Zeitstempeln«, Seite 123.

Quellen für digitale IDs. Digitale IDs können Sie von verschiedenen Quellen beziehen. Viele IDs sind für das Signieren von E-Mails vorgesehen; mit solchen IDs können Sie auch in PLOP DS PDF-Dokumente signieren. Von welcher Quelle Sie Ihre digitalen IDs beziehen, hängt von der benötigten Anzahl ab (z.B. eine ID pro Angestellter oder nur eine Unternehmens-ID) und vom gewünschten Grad an Kontrolle:

- Digitale ID von einer der öffentlichen Zertifizierungsstellen beziehen, die kommerzielle oder kostenlose IDs anbieten. Um die Signaturprüfung mit Acrobat zu erleichtern, empfehlen wir, Signaturen mit den digitalen IDs einer CA zu erstellen, die als vertrauenswürdige Stammzertifizierungsstelle *(Trusted Root)* in Acrobat installiert ist (siehe »Vertrauenswürdige Stammzertifikate in Acrobat (Trusted Root Certificates)«, Seite 93).
- Für große Unternehmen: Bauen Sie Ihre eigene private CA auf, um digitale IDs selbst erstellen zu können. Dazu sind verschiedene Softwarepakete auf dem Markt erhältlich, wie zum Beispiel die kostenlose OpenSSL-Software (siehe *www.openssl.org*), die in Java enthaltene Anwendung *keytool*, sowie die *Certificate Services*, die Bestandteil des Betriebssystems Microsoft Windows Server sind.
- Zu Testzwecken oder zum Austausch innerhalb einer kontrollierten oder kleinen Benutzergruppe: Erzeugen Sie eine digitale ID aus einem selbst signierten Zertifikat. In Acrobat gehen Sie dazu wie folgt vor:
  Acrobat XI/DC: *Bearbeiten, Voreinstellungen, Unterschriften, Identitäten und vertrauenswürdige Zertifikate, Weitere..., Digitale ID hinzufügen, Neue digitale ID, die ich jetzt erstellen möchte*;
  Acrobat X: *Werkzeuge, Signieren und zertifizieren, ...weitere Optionen, Sicherheitseinstellungen, Digitale IDs, Digitale ID hinzufügen, Neue digitale ID, die ich jetzt erstellen*

möchte;
Im nächsten Schritt können Sie eine PKCS#12-Datei auf der Festplatte oder den Zertifikatspeicher von Windows als Ziel angeben. Beide Methoden werden von PLOP DS unterstützt.

7.1.2 Signaturen in Acrobat und PDF

PDF unterstützt verschiedene Arten von digitalen Signaturen, die im Folgenden beschrieben werden. In PDF sind Signaturen als Formularfelder implementiert. PDF-Signaturen beziehen sich immer auf das gesamte Dokument (und nicht auf einzelne Seiten) und sind in zwei Varianten verfügbar:

- Unsichtbare Signaturen beanspruchen keinen Platz auf der Seite. Sie können in Acrobat im Navigationsfenster *Unterschriften* dargestellt werden (Acrobat X/XI/DC: *Anzeige, Ein-/Ausblenden..., Navigationsfenster, Unterschriften...).*
- Sichtbare Unterschriften verwenden ein rechteckiges Formularfeld, das an einer beliebigen Stelle auf einer Seite im Dokument positioniert ist. Sie können die Seitennummer, den Namen des Formularfelds und die Feldkoordinaten festlegen.

Für beide Arten von Signaturen können weitere Eigenschaften festgelegt werden, z.B. Ort, Grund für die Unterschrift oder Kontaktinformationen.

Hinweis *In früheren Acrobat-Versionen wurden gültige sichtbare Signaturen mit einem grünen Häkchen im Signaturfeld angezeigt. Wegen Verwirrung und potenziellen Problemen mit Fälschungen (z.B. könnten Benutzer eine bereits mit einem grünen Häkchen versehene »Visualisierungsseite« verwenden) entspricht dies nicht länger der empfohlenen Praxis. Statusinformationen sowie andere Signaturdetails für eine oder mehrere PDF-Signaturen werden auf der linken Seite im Navigationsfenster Unterschriften von Acrobat angezeigt, so dass das Signaturrechteck auf der Seite unverändert bleibt (unabhängig von der Gültigkeit der Signatur). Dies sorgt für eine deutlichere Darstellung des Signaturstatus für den Benutzer.*

Genehmigungssignaturen. Der am häufigsten für PDF verwendete Signaturtyp ist die Genehmigungssignatur. Ein PDF-Dokument kann ein oder mehrere davon enthalten. Eine Genehmigungssignatur wird in einem sichtbaren oder unsichtbaren Formularfeld vom Typ *Digitale Unterschrift* platziert. Sie gewährleistet, dass das Dokument vom Inhaber der digitalen ID signiert wurde und sorgt dafür, dass Änderungen am Dokument erkannt werden können. Durch eine Änderung am Dokument wird die Signatur ungültig. Genehmigungssignaturen beziehen sich auf eine Person oder Instanz, die die Signatur erstellt hat. Da niemand sonst Zugriff auf das erforderliche Kennwort oder die PIN hat, kann der Unterzeichner den Status des Dokuments zum Zeitpunkt der Unterschrift nicht verleugnen (Nichtabstreitbarkeit).

Wenn ein Dokument mit einer Genehmigungssignatur in Acrobat geöffnet wird, erscheint am oberen Fensterrand eine blaue Dokumentmitteilungsleiste (bei PDF/A-Konformität oder wenn das Dokument Formularfelder enthält, wird allerdings statt der Signaturinformationen der PDF/A-Status bzw. eine Meldung zum Formularfeld angezeigt). Ist die Signatur gültig, trägt die Mitteilungsleiste einen grünen Haken. Die Signatur wird auch im Navigationsfenster *Unterschriften* von Acrobat angezeigt.

Genehmigungssignaturen können optional Sperrinformationen zu Zertifikaten und einen Zeitstempel für die Langzeitvalidierung enthalten. Beide Elemente können von einem vertrauenswürdigen Server über das Netz bezogen werden, wenn die Signatur erzeugt wird.

Genehmigungssignaturen sind die Standardsignaturen in PLOP DS. Sie erfordern Ausgabe mit Typ PDF 1.6 oder höher. Wenn nötig, erhöht PLOP DS die PDF-Version entsprechend.

Genehmigungssignaturen werden in pCOS als *signaturefields[...]/sigtype=approval* ausgegeben.

Zertifizierungssignaturen. Bei der ersten Signatur in einem Dokument kann es sich um eine Zertifizierungssignatur handeln. Dieser Typ wird auch Autorensignatur genannt, weil der Status des Dokuments zertifiziert wird, so wie der Autor es erzeugt hat. Der Autor des Dokuments kann bestimmte Arten von Änderungen am Dokument zulassen, die die Signatur nicht ungültig machen. Zertifizierungssignaturen werden daher auch als MDP-Signaturen *(Modification Detection and Prevention)* bezeichnet. Die folgenden zulässigen Änderungsmöglichkeiten können angegeben werden (siehe Tabelle 7.6):

- Keine Änderungen erlaubt: nützlich für typische schreibgeschützte Dokumente wie Pressemitteilungen, Gesetzestexte usw. In diesem Fall wird die Zertifizierungssignatur sogar ungültig, wenn nur eine Genehmigungs- oder Zertifizierungssignatur hinzugefügt wird.
- Ausfüllen von Formularfeldern und Hinzufügen digitaler Signaturen (aber nur durch Anklicken eines Signaturfelds, nicht über das Acrobat-Menü): Zertifizierungssignaturen garantieren dem Benutzer, der das Formular ausfüllt, das es sich um das authentische Dokument, wie z.B. ein Bestellformular, handelt. Beim Ausfüllen von editierbaren Formularfeldern oder beim Anbringen einer Genehmigungssignatur wird die Zertifizierungssignatur nicht ungültig. Das Hinzufügen von Seiten durch Kopieren von Seiten-Templates ist ebenfalls erlaubt (nicht jedoch das manuelle Hinzufügen von Seiten), diese Methode wird jedoch selten verwendet.
- Ausfüllen von Formularfeldern, Hinzufügen digitaler Signaturen und Anmerkungen erlaubt: nützlich z.B. für einen Notar, der einen Kommentar mit Details zur Art der Bescheinigung zu einem signierten Dokument hinzufügen möchte.

Beim Öffnen eines Dokuments mit einer Zertifizierungssignatur wird in Acrobat am oberen Fensterrand in der Dokumentmitteilungsleiste eine Schleife angezeigt. Die Signatur wird in Acrobat auch im Navigationsfenster *Unterschriften* angezeigt (mit einer Schleife für eine gültige Signatur).

Zertifizierungssignaturen können in PLOP DS mit der Signaturoption *certification* erstellt werden (siehe Abschnitt 7.3.6, »Zertifizierungssignaturen«, Seite 113). Sie erfordern Ausgabe mit Typ PDF 1.6 oder höher. Wenn nötig, erhöht PLOP DS die PDF-Version entsprechend.

Zertifizierungssignaturen werden in pCOS als *signaturefields[...]/sigtype=certification* ausgegeben.

Zeitstempelsignaturen auf Dokumentebene. Zeitstempelsignaturen auf Dokumentebene dürfen nicht mit einem eingebetteten Zeitstempel in eine Genehmigungs- oder Zertifizierungssignatur verwechselt werden. Ein Dokument kann beliebig viele Zeitstempelsignaturen auf Dokumentebene enthalten. Zeitstempelsignaturen auf Dokumentebene garantieren, dass das Dokument zu einem bestimmten Zeitpunkt vorhanden war. Zeitstempel werden von einem vertrauenswürdigen Server über das Netz bezogen. Eine Zeitstempelsignatur bezieht sich nicht auf eine bestimmte Person oder Instanz, die das Dokument unterzeichnet hat. Zeitstempelsignaturen auf

Dokumentebene sind wichtig für die Langzeitvalidierung, weil damit bestehende Signaturen erneuert werden können. Sie werden in Formularfeldern platziert, sind aber immer unsichtbar.

Beim Öffnen eines Dokuments mit einer Zeitstempelsignatur wird in Acrobat am oberen Rand in der Dokumentmitteilungsleiste ein grüner Haken angezeigt. Die Signatur wird in Acrobat auch im Navigationsfenster *Unterschriften* angezeigt (mit einem Uhr-plus-Stempel-Symbol für eine gültige Signatur).

Zeitstempelsignaturen auf Dokumentebene können in PLOP DS mit der Signaturoption *doctimestamp* erstellt werden (siehe Abschnitt 7.5.3, »Zeitstempelsignaturen auf Dokumentebene«, Seite 125). Sie erfordern Ausgabe mit Typ PDF 1.7ext8 oder höher. Wenn nötig, erhöht PLOP DS die PDF-Version entsprechend.

Zeitstempelsignaturen werden in pCOS als *signaturefields[...]/sigtype=doctimestamp* ausgegeben.

Verwendungsrechtesignaturen. Ein Dokument kann bis zu zwei Verwendungsrechtesignaturen enthalten. Damit können Benutzern bestimmte Editierfunktionen in Adobe Reader ermöglicht werden; sie erhalten dadurch PDF-Dokumente mit erweiterter Reader-Funktionalität. Verwendungsrechtesignaturen sind nicht an Formularfelder gebunden und werden in Acrobat im Navigationsfenster *Unterschriften* nicht angezeigt.

Verwendungsrechtesignaturen können mit PLOP DS nicht erzeugt werden, können aber mit dem pCOS-Pseudo-Objekt *usagerights* abgefragt werden.

7.1.3 Vertrauenswürdige Stammzertifikate in Acrobat (Trusted Root Certificates)

Adobe Reader und Acrobat akzeptieren vertrauenswürdige Stammzertifikate aus den unten aufgeführten Quellen. Diese werden *Trusted Roots* oder *Trust Anchors* genannt. Mit *Bearbeiten, Voreinstellungen, Unterschriften, Identitäten und vertrauenswürdige Zertifikate, Weitere..., Vertrauenswürdige Zertifikate* können Sie die Liste der in Acrobat vertrauenswürdigen Stammzertifikate anzeigen lassen (siehe Abbildung 7.1). Zertifikate, die mit einem Zertifikat in der Liste der vertrauenswürdigen Stammzertifikate signiert sind, gelten als vertrauenswürdig. Da Acrobat zum erfolgreichen Validieren von Signaturen, die mit einem Zertifikat aus Acrobats vertrauenswürdigem Stammzertifikatspeicher signiert sind, vom Endbenutzer nicht konfiguriert werden muss, ist es empfehlenswert, Signaturen mit Zertifikaten einer Stammzertifizierungsstelle aus der AATL oder EUTL zu erstellen (siehe unten).

Adobe Approved Trust List (AATL). Die AATL[1] enthält Zertifizierungsstellen (CAs) aus Wirtschaft, Regierungsstellen und Institutionen in vielen Ländern der Welt. Bei Redaktionsschluss nahmen Dutzende von CAs am AATL-Programm teil.

AATL-Stammzertifikate sind in Acrobat und Adobe Reader X/XI/DC integriert. Acrobat vertraut diesen Stammzertifikaten sowie allen Zertifikaten, die mit einem dieser vertrauenswürdigen Stammzertifikate signiert sind. Um die Vertrauenswürdigkeit herzustellen, ist keine manuelle Konfiguration erforderlich. Die Liste kann in regelmäßigen Abständen automatisch aktualisiert werden oder manuell über *Bearbeiten, Vorein-*

1. Für weitere Informationen und eine Liste der teilnehmenden CAs siehe helpx.adobe.com/acrobat/kb/approved-trust-list1.html.

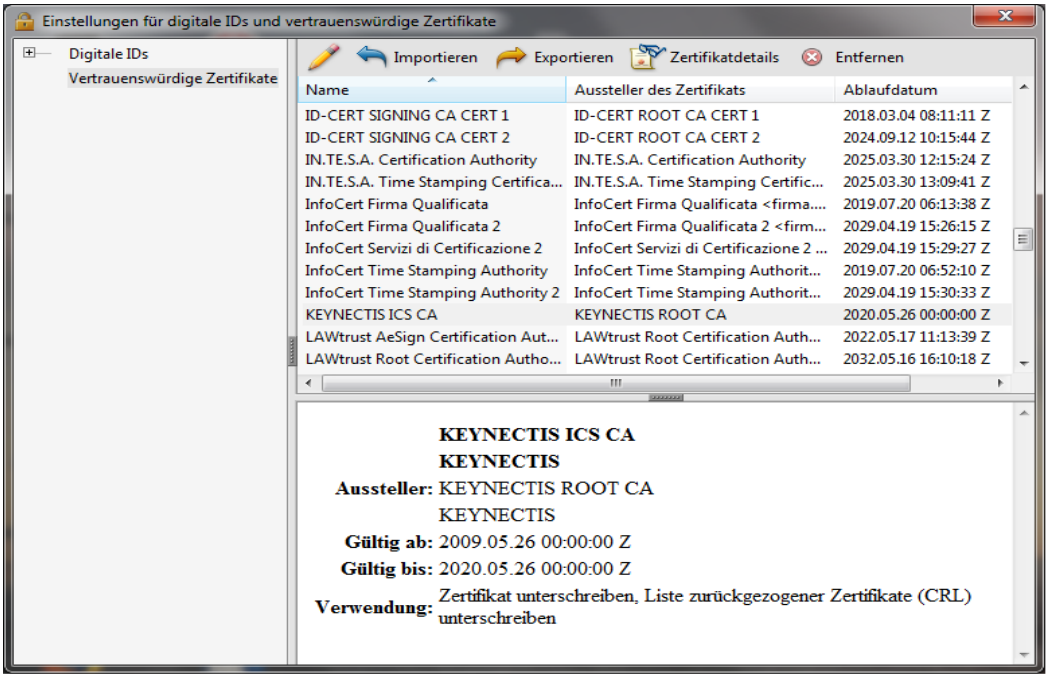


Abb. 7.1
Liste vertrauenswürdiger Zertifikate in Acrobat

stellungen, Berechtigungen, Automatisches Update der von Adobe geprüften und als vertrauenswürdig erachteten Zertifikate.

AATL-Zertifizierungsstellen stellen Zertifikate nur auf sicheren Tokens aus, die zertifiziert sind gemäß FIPS 140-2 Level 2 oder als *Secure Signature Creation Device* (SSCD) gemäß EU-Verordnung oder ähnlichen Standards. In vielen Fällen wird das Zertifikat auf einem SafeNet-Token oder einem Hardware-Security-Modul (HSM) gespeichert. AATL-Zertifikate werden nie als Datei, sondern nur auf einem sicheren Token gespeichert.

Einige Zertifizierungsstellen stellen sowohl AATL- als auch Nicht-AATL-Zertifikate unter dem selben Stamm aus. In diesem Fall muss in den Zertifikat-Richtlinien ausdrücklich darauf hingewiesen werden, dass das Zertifikat in Übereinstimmung mit den AATL-Regeln ausgestellt wurde. Andernfalls wird es von Acrobat unter den bekannten vertrauenswürdigen Stammzertifizierungsstellen als ungültig behandelt.

Acrobat enthält auch Stammzertifikate aus Adobes älterem Programm *Certified Document Services (CDS)* aus dem Jahr 2005, dem Vorgänger von AATL. Während AATL-Zertifizierungsstellen in Acrobat direkt als vertrauenswürdige Stammzertifizierungsstellen behandelt werden, sind CDS-Zertifikate vom Adobe-Stammzertifikat signiert. Folgende Zertifizierungsstellen sind Teil des CDS-Programms: Entrust, GlobalSign, Keynectis, Post.Trust und Symantec.

European Union Trust List (EUTL). Adobe Reader und Acrobat XI (ab 11.0.6) und DC unterstützen vertrauenswürdige Stammzertifikate der *European Union Trust List* (EUTL) gemäß ETSI TS 119 612[1]. Das EUTL-Update lässt sich folgendermaßen in Acrobat einrichten:

1. Siehe www.etsi.org/deliver/etsi_ts/119600_119699/119612/01.01.01_60/ts_119612v010101p.pdf und ec.europa.eu/digital-agenda/en/eu-trusted-lists-certification-service-providers

Bearbeiten, Voreinstellungen, Berechtigungen, Automatische Updates der von der Europäischen Union geprüften und als vertrauenswürdig erachteten Zertifikate. Die ersten EUTL-Updates wurden 2015 für den Download in Acrobat zur Verfügung gestellt. EUTL umfasst Stammzertifikate der *Trusted Lists* aller EU-Mitgliedstaaten gemäß eIDAS-Framework (EU-Verordnung 910/2014).

Hinzufügen von vertrauenswürdigen Stammzertifikaten. Acrobat akzeptiert auch manuell in Acrobat oder Adobe Reader importierte Stammzertifikate über *Bearbeiten, Voreinstellungen, Unterschriften, Identitäten und vertrauenswürdige Zertifikate, Weitere..., Vertrauenswürdige Zertifikate*. Das Zertifikat muss als vertrauenswürdiges Stammzertifikat konfiguriert werden: klicken Sie auf *Einstellungen für Vertrauenswürdigkeit bearbeiten* und aktivieren Sie im Register *Vertrauenswürdigkeit* den Eintrag *Dieses Zertifikat als vertrauenswürdigen Stamm verwenden*. Dies kann nützlich sein für Unternehmens-PKIs mit benutzerdefinierten Stammzertifizierungsstellen. Diese Konfiguration ist zwar für jedes beliebige vertrauenswürdige Stammzertifikat möglich, erfordert allerdings Benutzerinteraktion, was in manchen Arbeitsabläufen unerwünscht ist.

Zertifikate im Zertifikatspeicher von Windows. Acrobat behandelt optional Zertifikate im Zertifikatspeicher von Windows als vertrauenswürdig. Dies kann über *Bearbeiten, Voreinstellungen, Unterschriften, Überprüfung, Weitere..., Windows-Integration* eingerichtet werden.

7.2 Signieren von Dokumenten mit PLOP DS

7.2.1 Überblick

PLOP DS unterstützt mehrere kryptografische Engines, die Public-Key- und Hash-Algorithmen implementieren, die für die digitale Signatur eines Dokuments erforderlich sind. Signaturen werden in der PLOP DS-Bibliothek mit den API-Funktionen *PLOP_prepare_signature()* und *PLOP_create_document()* vorbereitet. Für das PLOP-Kommandozeilen-Tool verwenden Sie die Option *--signopt* (Kurzform: *-S)*.

Um eine digitale Signatur mit PLOP DS zu verwenden, benötigen Sie eine digitale ID. Wenn Sie mit einer Datei für digitale IDs oder einem Token arbeiten, benötigen Sie das zugehörige Kennwort. Bei Verwendung einer persönlichen oder kontospezifischen digitalen ID im Zertifikatspeicher von Windows ist die ID normalerweise durch die Windows-Anmeldung geschützt.

Krypto-Engines zur Erstellung digitaler Signaturen. PLOP DS unterstützt verschiedene kryptografische Engines. Bei einer kryptografischen Engine handelt es sich um Soft- oder Hardware, die die erforderlichen kryptografischen Funktionen zur Erzeugung einer digitalen Signatur implementiert. Die Wahl einer kryptografischen Engine bestimmt Format und Speicherort der digitalen IDs sowie die Integration mit anderer Software und dem Betriebssystem. PLOP DS unterstützt die folgenden kryptografischen Engines:

- Die interne Engine *builtin* implementiert die erforderlichen kryptografischen Funktionen direkt im Kern von PLOP DS ohne externe Abhängigkeiten. Sie ist standardmäßig aktiviert, kann aber auch explizit mit der Option *engine=builtin* ausgewählt werden.
- Die *pkcs#11*-Engine bezieht sich auf die PKCS#11-Software-Schnittstelle, die einen einheitlichen Zugang für kryptografische Tokens bietet, wobei Token hier für Smartcard, USB-Stick oder andere kryptografische Geräte steht. Tokens bieten eine höhere Sicherheit als Softwarezertifikate und sind oft durch eine PIN geschützt. Die PKCS#11-Engine kann auch für den Zugriff auf ein Hardware-Security-Modul (HSM) verwendet werden. Die *PKCS#11*-Engine lässt sich mit der Signaturoption *engine=pkcs#11* auswählen.
- Die *mscapi*-Engine bezieht sich auf das *Microsoft Cryptographic API* (nur unter Windows verfügbar), das Bestandteil des Betriebssystems ist. Damit kann PLOP DS sowohl die kryptografische Infrastruktur von Windows nutzen als auch Soft- oder Hardware von Drittanbietern, auf die über einen CAPI-Treiber zugegriffen werden kann. Die *mscapi*-Engine lässt sich mit der Signaturoption *engine=mscapi* auswählen.
- Alternativ kann eine vom Benutzer bereitgestellte Krypto-Engine verwendet werden, um sicherzustellen, dass alle Verschlüsselungsoperationen (Hashing und Signieren) in einer dedizierten Krypto-Bibliothek durchgeführt werden. Die Anbindung eines solchen externen Verschlüsselungsmoduls erfordert einen speziellen PLOP-Build, der auf Anfrage erhältlich ist.

Unterstützte Formate für digitale IDs. PLOP DS benötigt zum Signieren eines PDF-Dokuments eine digitale ID. Sie enthält das digitale Zertifikat des Unterzeichners sowie den zugehörigen privaten Schlüssel und ist normalerweise durch ein Kennwort oder auf eine andere Art geschützt. PLOP DS unterstützt folgende Arten von digitalen IDs:

- Plattformen mit *engine=builtin:* digitale IDs in einer Datei im PKCS#12-Format (in der Regel *.p12* oder *.pfx)*
- Plattformen mit PKCS#11-Unterstützung mit *engine=pkcs#11:* digitale IDs auf einer Smartcard oder einem anderen, am Computer angeschlossenen kryptografischen Token (Gerät).
- Windows mit *engine=mscapi:* digitale IDs im Zertifikatspeicher von Windows.

7.2.2 Signieren mit der integrierten Engine

Die interne Engine ist die Standard-Engine. Sie kann für dateibasierte digitale IDs verwendet werden und bietet die volle Funktionalität und alle Möglichkeiten zur Steuerung.

Entsperren des privaten Schlüssels. Digitale IDs (genauer: der private Schlüssel in der digitalen ID) sind im Allgemeinen durch ein Kennwort, eine Passphrase oder PIN geschützt, da sie den vertraulichen privaten Schlüssel zur Erzeugung der digitalen Signatur enthalten. Um eine digitale ID für PLOP DS zu entsperren, benötigen Sie die korrekte Authentisierung. Wenn Sie ein falsches Kennwort übergeben, löst PLOP DS eine Exception aus.

Sie müssen das passende Kennwort mit der Signaturoption *password* übergeben. Für das PLOP DS-Kommandozeilen-Tool sollten Sie das Kennwort unbedingt indirekt in einer Hilfsdatei mit der Unteroption *passwordfile* übergeben. Bei direkter Übergabe des Kennworts ohne Datei könnten andere es sonst möglicherweise lesen, da die Kommandozeile auf einem Mehrbenutzer-System für andere Benutzer sichtbar sein kann.

Beispiel für Optionslisten. Das Beispiel unten zeigt, wie man PDF-Dokumente mit dem PLOP DS-Kommandozeilen-Tool digital signiert. Die mit *--signopt* übergebene Optionsliste kann an die API-Funktion *PLOP_prepare_signature( )* übergeben werden, um eine Signatur mit Ihrem eigenen Programm zu erstellen. Ausführliche Programmierbeispiele für alle unterstützten Sprachbindungen sind im PLOP DS-Paket enthalten. In den Beispielen wird eine digitale ID in der Datei *demo_signer_rsa_2048.p12* mit dem Kennwort *demo* verwendet, die in den PLOP DS-Paketen enthalten ist.

Eine unsichtbare Signatur für ein PDF-Dokument mit Hilfe einer digitalen ID aus der Datei *demo_signer_rsa_2048.p12* erstellen. Das Kennwort für die digitale ID befindet sich in der Datei *pw.txt*:

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt" ↵
       --outfile signed.pdf input.pdf
```

7.2.3 PKCS#11-Engine für einen kryptografischen Token

Mithilfe der PKCS#11-Engine in PLOP DS können Sie Zertifikate auf einem kryptografischen Token, wie z.B. einer Smartcard oder einem USB-Stick, oder auf einem Hardware-Security-Modul (HSM) verwenden. Zur Signaturerstellung mit einem solchen Gerät benötigen Sie eine DLL oder eine dynamische Bibliothek, die ein tokenspezifisches Protokoll implementiert. Diese PKCS#11-DLL/SO wird vom Token-Hersteller als Teil des zugehörigen Softwarepakets bereitgestellt. Sie muss auf dem System installiert und PLOP DS verfügbar gemacht werden. Unter Windows muss die DLL dazu entweder in das Windows-Systemverzeichnis, ein Verzeichnis in der Umgebungsvariablen PATH oder in das aktuelle Verzeichnis der Anwendung kopiert werden. Beachten Sie, dass eine PKCS#11-

Abb. 7.2
Smartcard-Reader mit Tastatur (links) und kryptografischer USB-Token (rechts). Beide Geräte können mit PLOP DS über die PKCS#11-Engine verbunden werden.

DLL/SO von anderen DLLs abhängig sein kann. In diesem Fall müssen alle vom Hersteller bereitgestellten DLLs für PLOP DS verfügbar sein.

Auswahl eines privaten Schlüssels. Ein Signaturgerät kann verschiedene digitale IDs enthalten, z.B. eine zum Entschlüsseln von E-Mails und eine andere für das digitale Signieren von Dokumenten. Gibt es auf dem Token mehrere Signaturzertifikate, müssen Sie eine der Unteroptionen *issuer*, *label*, *serial* oder *subject* der Option *digitalid* verwenden, um das gewünschte Zertifikat anhand eines dieser Kriterien auszuwählen. Mit Hilfe der Verwaltungssoftware für den Token kann ein Schlüssel mit einem Label versehen werden. Aussteller, Seriennummer und Betreff *(subject)* sind interne Felder von Zertifikaten.

Entsperren des privaten Schlüssels auf einem Token. Erlaubt ein kryptografisches Token die Übergabe eines Kennworts oder einer PIN über die Software, müssen Sie wie bei *engine=builtin* die Signaturoption *password* übergeben. Muss das Kennwort oder die PIN direkt eingegeben werden (z.B. bei einem Smartcard-Reader mit Tastatur), können Sie die Option *password* weglassen (oder einen leeren String übergeben) und müssen die PIN über die Tastatur des Tokens manuell eingeben. Die Details der Kennwort- bzw. PIN-Verarbeitung hängen vom Typ des kryptografischen Tokens ab.

Einige Tokens melden sich nach einer bestimmten Zeit oder einer bestimmten Anzahl von Unterschriften automatisch ab. Für Massensignaturen müssen Sie den Token entsprechend konfigurieren, um automatische Abmeldung zu vermeiden. Weitere Informationen finden Sie in Ihrer Token-Dokumentation. Bei der automatischen Abmeldung des Tokens wird folgende Fehlermeldung ausgegeben:

```
Error adding signature data ('PKCS#11: couldn't create signature
(C_Sign: CKR_USER_NOT_LOGGED_IN)')
```

PKCS#11-Beispiele. In den folgenden Beispielen wird die herstellerspezifische PKCS#11-DLL *cryptoki.dll* genannt. Der Name der tatsächlichen DLL kann davon abweichen.

Erstellen einer unsichtbaren Signatur für ein PDF-Dokument mit Hilfe einer digitalen ID von einem Token, das über *PKCS#11* angesprochen wird. Die PIN für den Token befindet sich in der Datei *pw.txt*:

```
plop --signopt "engine=pkcs#11 digitalid={filename=cryptoki.dll} passwordfile=pw.txt" ↵
       --outfile signed.pdf input.pdf
```

Erstellen einer unsichtbaren Signatur für ein PDF-Dokument mit Hilfe einer digitalen ID von einem Token, der über *PKCS#11* angesprochen wird. In diesem Kommando wird keine PIN übergeben; die PIN für den Token muss stattdessen über die integrierte Token-Tastatur eingegeben werden:

```
plop --signopt "engine=pkcs#11 digitalid={filename=cryptoki.dll}" ↵
       --outfile signed.pdf input.pdf
```

7.2.4 PKCS#11-Engine für ein Hardware-Security-Modul (HSM)

Ein Hardware-Security-Modul (HSM) bietet hardwarebasierte Sicherheit für den privaten Schlüssel und gegenüber Tokens oder Smartcards eine erhebliche Leistungssteigerung. HSMs werden typischerweise in folgenden Szenarien eingesetzt:

- Kommerzielle Zertifizierungsstellen bieten HSM-basierte Zertifikate an. Ein HSM wird typischerweise von einer Zertifizierungsstelle bereitgestellt und verwaltet. Für anspruchsvolle Anwendungen kann sich das HSM am Standort des Kunden befinden. GlobalSign, QuoVadis und Symantec bieten z.B. HSM-basierte AATL-Zertifikate.
- Bereitstellung eines HSM mit einer Unternehmens-PKI vor Ort.
- Bereitstellung in der Cloud: HSM-Dienste können zusammen mit CPU- und Speicherdiensten gebucht werden. Amazon Web Services (AWS) und Microsoft Azure bieten HSM-Hosting.

Bei unseren Tests haben wir festgestellt, dass die komplette HSM-Leistung nur mit einer Multi-Thread-Client-Anwendung oder vielen unabhängigen Clients gleichzeitig erreicht werden kann.

Beispiel für PKCS#11-HSM. Das folgende Beispiel zeigt digitale Signaturen mit einem Thales nShield HSM. Wir gehen davon aus, dass für ein mit dem HSM erzeugtes Schlüsselpaar ein passendes Zertifikat vorhanden ist und eine geeignete Sicherheitsumgebung mit der Verwaltungssoftware konfiguriert wurde, mit der das nShield-Gerät ausgestattet ist (für weitere Informationen siehe die nShield-Dokumentation).

Sobald diese Vorbereitungen abgeschlossen sind, kann PLOP DS mit folgendem Aufruf dazu verwendet werden, PDF-Dokumente zu signieren:

```
plop --signopt "engine=pkcs#11 digitalid={label=demo_signer_rsa_2048 ↵
       filename=/opt/nfast/toolkits/pkcs11/libcknfast.so}" -o signed.pdf input.pdf
```

PKCS#11-Sessions und Multithreading. Um den Durchsatz bei Massensignaturen zu steigern, minimiert PLOP DS die Anzahl der Operationen für das Laden/Entladen für die PKCS#11-DLL/SO und maximiert die Dauer jeder PKCS#11-Session. Die Anwendung muss dazu folgende Bedingungen erfüllen:

- Bis *PLOP_delete()* für das letzte PLOP-Objekt aufgerufen wurde, das die Bibliothek verwendet, darf gleichzeitig nicht mehr als eine PKCS#11-DLL/SO geladen werden. Nachdem das letzte PLOP-Objekt gelöscht wurde, darf eine weitere PKCS#11-DLL/SO in *PLOP_prepare_signature()* angegeben werden. Mit anderen Worten, jede beliebige

Abb. 7.3
Ein Hardware-Security-Modul (HSM) kann über die PKCS#11-Engine mit PLOP DS verbunden werden.

Anzahl von PKCS#11-Slots kann im Multithread-Verfahren angesprochen werden, sofern alle Token-Slots durch die selbe DLL/SO bedient werden (was normalerweise für Tokens des selben Typs gilt).

- *PLOP_prepare_signature()* in einem bestimmten Thread darf auf keinen PKCS#11-Slot zugreifen, auf den bereits von einem anderen Thread aus zugegriffen wird. Multithread-Anwendungen, die innerhalb von mehreren Threads mit dem gleichen Token signieren möchten, müssen die Threads mit geeigneten Mitteln synchronisieren, z.B. mithilfe eines Mutex.
- Für Multithread-Anwendungen ist eine thread-sichere PKCS#11-Bibliothek erforderlich. Mit der Unteroption *threadsafe=true* der Option *digitalid* lässt sich prüfen, ob die Bibliothek thread-sicher ist, und diese als thread-sicher initialisieren.
- Die PKCS#11-DLL/SO kann optional mit der Unteroption *sticky* der Signaturoption *digitalid* auch nach der letzten Verwendung im Speicher belassen werden. Dies kann den Signaturvorgang leicht beschleunigen. Es kann dann jedoch keine andere PKCS#11-DLL/dynamische Bibliothek im selben Prozess geladen werden.

Eine neue Session wird im ersten Aufruf von *PLOP_prepare_signature()* für einen bestimmten Slot erzeugt und aufrecht erhalten, bis *PLOP_prepare_signature()* erneut im selben Thread aufgerufen wird. Werden in einem Thread keine weiteren Signaturen mehr erzeugt, kann die PKCS#11-Session explizit mit der Option *signature=false* von *PLOP_prepare_signature()* beendet werden. Deshalb sollte die Anwendung *PLOP_prepare_signature()* nur einmal für so viele Ausgabedokumente wie möglich aufrufen. Solange zum Beispiel der selbe PKCS#11-Slot adressiert wird und die Beschränkungen des Tokens erfüllt sind (z.B. maximale Anzahl von Signaturen oder maximale Zeit für aufeinanderfolgende Signaturen), sind keine weiteren Aufrufe von *PLOP_prepare_signature()* mehr erforderlich.

Ein vollständiges Codebeispiel zur effizienten Erstellung von Massensignaturen finden Sie im Beispiel *multisign*, das Bestandteil aller PLOP DS-Pakete ist. Beachten Sie, dass die *multisign*-Logik im Vergleich zum PLOP DS-Kommandozeilen-Tool signifikante Leistungsvorteile für tokenbasierte Signaturen bietet.

7.2.5 Signieren mit der MSCAPI-Engine unter Windows

Mit der MSCAPI-Engine können Sie die ins Windows-Betriebssystem integrierten Signaturfunktionen nutzen. Vor allem können Sie auf die digitalen IDs im Zertifikatspeicher von Windows zugreifen. Die MSCAPI-Engine hat jedoch auch einige Beschränkungen, von denen andere kryptografische Engines nicht betroffen sind. Beispielsweise unterstützt MSCAPI kein ECDSA.

Hinweis *Die Einbettung von OSCPs und CRLs sowie Zeitstempel werden für* `engine=mscapi` *nicht unterstützt. Deshalb können mit der MSCAPI-Engine keine LTV-fähigen Signaturen erstellt werden.*

Entsperren des privaten Schlüssels. Abhängig von Ihren Zertifikatseinstellungen können die digitalen IDs im Zertifikatspeicher von Windows durch Ihre Windows-Anmeldung geschützt sein, so dass kein zusätzliches Kennwort benötigt wird. Falls Sie unter Windows beim Import des Zertifikats erhöhte Sicherheit eingestellt haben, werden Sie jedes Mal, wenn das Zertifikat zum Signieren verwendet wird, zur Eingabe des Kennworts aufgefordert.

Beispiele für MSCAPI-Optionslisten. Die Beispiele unten gehen davon aus, dass die digitalen IDs zum Signieren im Zertifikatspeicher von Windows verfügbar sind. Um dies mit den Demo-Zertifikaten von PLOP DS zu erreichen, müssen Sie die digitale ID in der Datei *demo_signer_rsa_2048.p12* mit einem Doppelklick im Zertifikatspeicher von Windows installieren.

Erstellen einer unsichtbaren Signatur für ein PDF-Dokument mit Hilfe eines Zertifikats aus dem Zertifikatspeicher von Windows (aus dem Standardspeicher *My* und dem Standard-Speicherort *current_user)*. Das Beispiel geht davon aus, dass die digitale ID durch die Windows-Anmeldung geschützt ist, so dass kein Kennwort benötigt wird:

```
plop --signopt "engine=mscapi digitalid={store=My subject= ↵
        {PLOP Demo Signer RSA-2048}}"  --outfile signed.pdf input.pdf
```

Erstellen einer unsichtbaren Signatur für ein PDF-Dokument mit Hilfe einer digitalen ID aus der Datei *demo_signer_rsa_2048.p12*:

```
plop --signopt "engine=mscapi digitalid={filename=demo_signer_rsa_2048.p12} ↵
        passwordfile=pw.txt"  --outfile signed.pdf input.pdf
```

Erstellen einer unsichtbaren Signatur und Verschlüsseln des Dokuments mit dem Master-Kennwort *SECRET* für PDF-Verschlüsselung und mit dem Kennwort *demo* für den Zugriff auf digitale IDs:

```
plop --master SECRET --signopt "digitalid={filename=demo_signer_rsa_2048.p12} ↵
        password={demo}" --outfile signed.pdf input.pdf
```

Verwalten des Zertifikatspeichers von Windows. Zertifikate können unter Windows in verschiedenen Zertifikatspeichern verwaltet werden. Um ein neues Zertifikat im PKCS#12-Format zu installieren, doppelklicken Sie darauf und folgen dem Zertifikatimport-Assistenten. Sie können dies mit den Demo-Zertifikaten im PLOP DS-Paket unter Verwendung des Kennworts *demo* ausprobieren.

Sie können Zertifikate mit der Microsoft Management Console (MMC) wie folgt anzeigen und organisieren:

- Klicken Sie auf *Start* und geben Sie im Eingabefeld für Programmnamen *mmc* ein, um das Programm zu starten.
- Klicken Sie im Menü *Datei* auf *Snap-in hinzufügen/entfernen...*
- Unter *Verfügbare Standalone Snap-Ins* wählen Sie *Zertifikate* und klicken auf *Hinzufügen*.
- Im nächsten Dialog *Zertifikat-Snap-In* wählen Sie *Eigenes Benutzerkonto* und dann *Fertigstellen*. Wählen Sie alternativ *Dienstkonto* oder *Computerkonto*, falls sich Ihre Zertifikate dort befinden.
- Klicken Sie auf *OK*.

Jetzt können Sie die installierten Zertifikate durchsuchen. Ihre eigenen Zertifikate finden Sie in der Kategorie *Persönlich*, die in PLOP DS mit der folgenden Optionsliste adressiert werden kann (entweder übergeben an die Kommandozeilenoption *--signopt* oder an *PLOP_prepare_signature()*):

```
engine=mscapi digitalid={store=My subject={PLOP Demo Signer RSA-2048}}
```

Zertifikatdetails können Sie in der Management Console durch Doppelklick auf ein Zertifikat anzeigen lassen. Um ein Zertifikat im PFX-Format zu exportieren, rechtsklicken Sie auf ein Zertifikat in der Liste und wählen Sie *Alle Aufgaben, Exportieren...*. Der Zertifikatexport-Assistent wird gestartet.

Mit der Management Console können Sie ein Zertifikat auch importieren: rechtsklicken sie auf einen Zertifikatspeicher, z.B. *Persönlich* und wählen Sie *Alle Aufgaben, Importieren...* .

7.2.6 Kryptografische Details

Digitale Signaturen sind durch einen Verschlüsselungsalgorithmus und einen Hash-Algorithmus sowie Parameter für beide gekennzeichnet. Verschlüsselungsalgorithmus und Schlüssellänge für die Erzeugung der Signatur sind abhängig von der digitalen ID des Unterzeichnenden. Sie werden bei der Erzeugung des öffentlichen/privaten Schlüsselpaares für die digitale ID festgelegt. PLOP DS unterstützt die unten aufgeführten Signaturalgorithmen.

RSA-Signaturen. RSA wird unterstützt mit einer Schlüssellänge im Bereich 1024-8192 (2048-Bit oder mehr empfohlen). RSA ist im Internet und vielen anderen Anwendungsbereichen weit verbreitet. RSA-Signaturen erfordern eine so genannte Encoding-Methode *(Encoding Method for Signatures with Appendix, EMSA):*

- Die Standard-Encoding-Methode gemäß PKCS#1 v1.5 wird in allen Acrobat-Versionen unterstützt. Allerdings wird sie in vielen Signaturanwendungen demnächst auslaufen.
- Die neuere Encoding-Methode EMSA-PSS (*Probabilistic Signatur Scheme)* gemäß RFC 3447/RFC8017 bietet nachweisbare Sicherheit, erfordert für die Validierung jedoch Acrobat XI 11.0.19 oder Acrobat DC 2015.006.30280 für Windows (Aktualisierung im Januar 2017) oder höher. Ältere Versionen von Acrobat betrachten PSS-Signaturen als ungültig. Acrobat für OS X/macOS unterstützt bislang noch keine PSS-Signaturen. EMSA-PSS-Signaturen lassen sich mit der Signaturoption *rsaencoding=pss* erzeugen.

Hinweis EMSA-PSS-Signaturen werden nur für `engine=builtin` *unterstützt.*

DSA-Signaturen. DSA wird unterstützt mit einer Schlüssellänge im Bereich 1024-4096 (2048-Bit oder mehr empfohlen). DSA wird selten verwendet. Da Acrobat nur DSA mit dem unsicheren Hash-Algorithmus SHA-1 unterstützt, gibt es Sicherheitsbedenken bei der Verwendung von DSA.

Hinweis DSA-Signaturen werden für `engine=pkcs#11` *nicht unterstützt.*

Elliptic Curve Cryptography. ECDSA *(Elliptic Curve Digital Signature Algorithm)* ist der moderne Nachfolger von RSA. Im Allgemeinen gehen die Schlüssellängen bis zu 512-Bit (224-Bit oder mehr empfohlen). Mit ECDSA kann im Vergleich zu RSA mit kürzeren Schlüsseln die gleiche kryptografische Stärke erreicht werden, was wiederum die Leis-

tung verbessert. Die Stärke von ECDSA wird durch eine Kurve bestimmt, die durch Parameter oder häufiger einen Namen gekennzeichnet wird. Es gibt drei gängige Gruppen von ECDSA-Kurven:

- Die gängigsten Kurven wurden vom NIST standardisiert und sind in RFC 5480 aufgeführt. Diese sind *P-256*, *P-384* und *P-521*; andere Namen für diese Kurven sind *secp256r1* (oder *prime256v1*), *secp384r1* und *secp521r1*. Diese Kurven werden in Acrobat XI/DC unterstützt.
- RFC 5480 definiert eine zusätzliche Menge von 12 benannten, vom NIST empfohlenen Kurven. Diese werden auch in Acrobat XI/DC unterstützt, wenn die digitale ID direkt in Acrobat geladen wird, jedoch nicht für Zertifikate im Zertifikatspeicher von Windows oder dem Schlüsselbund von OS X/macOS.
- RFC 5639 definiert eine Reihe von Kurven, die sogenannten Brainpool-Kurven. Auf Brainpool-Kurven basierende Signaturen können nicht mit Acrobat XI/DC validiert werden. Leider zeigt Acrobat nicht klar an, dass der Signaturalgorithmus nicht unterstützt wird, sondern gibt folgende Fehlermeldung für Brainpool-Signaturen aus:

  ```
  Die Formatierung oder in dieser Unterschrift enthaltene Informationen weisen Fehler auf.
  ```

Da Acrobat auf Brainpool basierende Kurven nicht validieren kann, erfordern diese die Signaturoption *conformance=extended*.

Hinweis *ECDSA-Signaturen werden für* `engine=mscapi` *nicht unterstützt.*

Hash-Algorithmen. Ein Hash-Algorithmus wird verwendet, um einen *Message Digest* für die signierten Daten zu erzeugen. Verbreitete Hash-Algorithmen sind SHA-1 (gilt inzwischen als unsicher) und die stärkeren Algorithmen der SHA-2-Familie, die SHA-256, SHA-384 und SHA-512 umfasst. Der Hash-Algorithmus für eine Signatur lässt sich in Acrobat XI/DC folgendermaßen anzeigen (siehe Abbildung 7.4):

- öffnen Sie das Navigationsfenster *Unterschriften*;
- wählen Sie eine Signatur und dann *Unterschrifteneigenschaften einblenden...* im Menü *Unterschriften*.
- klicken Sie auf *Erweiterte Eigenschaften*;
- der resultierende Dialog *Erweiterte Unterschriftseigenschaften* zeigt die *Unterschriftsdetails* einschließlich des Hash-Algorithmus.

Tabelle 7.1 listet Signaturalgorithmen und zugehörige Hash-Funktionen auf. In der Tabelle ist auch die minimal erforderliche Acrobat-Version für die Validierung einer Signatur aufgeführt. Wenn Sie PDF-Signaturen mit Acrobat validieren wollen, müssen Sie eine Acrobat-Version verwenden, die die Signatureigenschaften auch unterstützt. Ta-

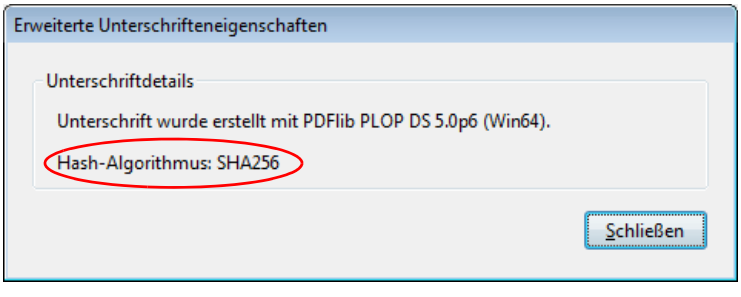


Abb. 7.4
Acrobat zeigt den für eine Signatur verwendeten Hash-Algorithmus an

belle 7.1 zeigt auch die minimale Version der PDF-Ausgabe, die für jeden Signaturalgorithmus erzeugt wird. Verwendet das Eingabedokument eine niedrigere PDF-Version, erhöht PLOP DS die PDF-Version des Ausgabedokuments auf die in der Tabelle angegebene Version.

Tabelle 7.1 Signaturalgorithmen, Hash-Algorithmen, Version der PDF-Ausgabe und erforderliche Acrobat-Versionen

Signaturalgorithmus	***Hash-Algorithmus***	***Version der PDF-Ausgabe und minimale Acrobat-Version für die Validierung[1]***
Genehmigungs- und Zertifizierungssignaturen		
RSA bis zu 8192 Bit	*SHA-256*	`sigtype=cades`:*PDF 1.7ext8 / Acrobat X* `sigtype=cms`: *PDF 1.6 / Acrobat 7*[2] `rsaencoding=pss`: *Acrobat XI 11.0.19 oder Acrobat DC 2015.006. 30280 (Januar 2017) für Windows*
DSA bis zu 4096 Bit	*SHA-1 (Acrobat X/XI/DC unterstützen keine anderen Hash-Algorithmen für DSA)*	`sigtype=cades`: *PDF 1.7ext8 / Acrobat X* `sigtype=cms`: *PDF 1.6 / Acrobat 7*
ECDSA mit NIST-Kurven (RFC 5480) P-256/P-384/P-521	*SHA-256, SHA-384 oder SHA-512, abhängig von der Kurve*	*PDF 1.7ext8 / Acrobat XI*
ECDSA mit NIST-Kurven (RFC 5480) außer P-256/P-384/P-521	*SHA-256, SHA-384 oder SHA-512, abhängig von der Kurve*	*PDF 1.7ext8 / (nicht mit dem Zertifikatspeicher von Windows oder dem Schlüsselbund von OS X/macOS)*
ECDSA mit 14 Brainpool-Kurven (RFC 5639)	*SHA-256, SHA-384 oder SHA-512, abhängig von der Kurve*	*PDF 1.7ext8 / kann nicht mit Acrobat XI/DC validiert werden* *(erfordert* `conformance=extended`)
Zeitstempel auf Dokumentebene		
von der TSA festgelegt	*standardmäßig SHA-256, lässt sich aber mit der Unteroption* `hash` *der Option* `doctimestamp` *ändern*	*PDF 1.7ext8 mit Erweiterungen gemäß PAdES Teil 4 / Acrobat X*
OCSP-Anfrage und -Antwort (Zertifikatidentifikation)		
vom OCSP-Responder festgelegt	*standardmäßig SHA-1, lässt sich aber mit der Unteroption* `hash` *der Option* `ocsp` *ändern*	*Acrobat DC und früher unterstützen nur SHA-1 für OCSP;* *(andere erfordern* `conformance=extended`)

1. Im PDF/A- und PDF/X-Modus bleibt die PDF-Version des Eingabedokuments unverändert.
2. RSA-8192-Schlüssel erfordern für die Validierung Acrobat X oder höher und werden von Acrobat unter OS X/macOS nicht unterstützt.

7.3 PDF-Aspekte von Signaturen

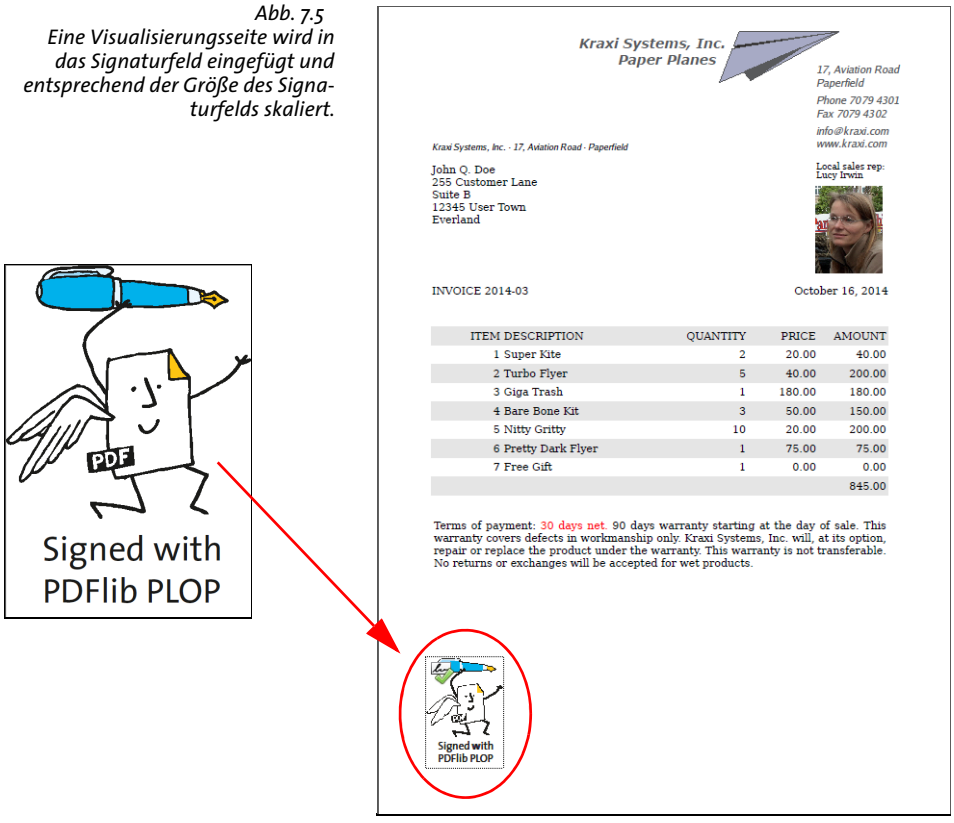
7.3.1 Visualisieren von Signaturen mit Grafik oder Logo

Digitale Signaturen können in einem Dokument folgendermaßen dargestellt werden:

- Unsichtbare Signaturen haben keine Darstellung auf der Seite. Sie werden nur im Navigationsfenster *Unterschriften* von Acrobat angezeigt. Zeitstempelsignaturen auf Dokumentebene werden immer als unsichtbare Signaturen erzeugt.
- Sichtbare Signaturen können beliebigen Text oder Grafiken enthalten, um die Signatur an einer bestimmten Stelle auf einer Seite darzustellen (siehe Abbildung 7.5). Eine Seite aus einem bestehenden PDF-Dokument kann für die Visualisierung einer Signatur verwendet werden. Sichtbare Signaturen werden ebenfalls im Navigationsfenster *Unterschriften* von Acrobat dargestellt. Das Dokument, aus dem die Seite entnommen wird, heißt Visualisierungsdokument. Da die Visualisierungsseite im Signaturfeld platziert wird, können Sie auf die Visualisierung im signierten Dokument klicken, um die Signatur in Acrobat zu validieren.

Visualisierungsdokument für Signaturen. Die PDF-Seite für die Signaturvisualisierung kann eine gescannte manuelle Unterschrift, ein offizielles Siegel oder Firmenlogo, ein Foto vom Inhaber des Signaturzertifikats oder eine andere sichtbare Darstellung enthalten, die für die Empfänger des signierten Dokuments nützlich ist.

Abb. 7.5
Eine Visualisierungsseite wird in das Signaturfeld eingefügt und entsprechend der Größe des Signaturfelds skaliert.



Verwendet das Visualisierungsdokument eine höhere PDF-Version als das signierte Eingabedokument, wird die PDF-Version der erzeugten Ausgabe entsprechend angepasst. Dokumente für *PDF 1.7ext 3* (Acrobat 9) und *PDF 1.7ext8* (Acrobat X/XI/DC) sind hinsichtlich ihrer Verwendung einer Visualisierungsseite kompatibel zu PDF 1.7.

Hinweis *Bei PDF/A-Signaturen unterliegt das Visualisierungsdokument einer Reihe von Bedingungen (siehe »PDF/A-Konformität«, Seite 107). Visualisierung von Signaturen wird im PDF/X- und PDF/VT-Modus nicht unterstützt.*

Das Visualisierungsdokument muss mit *PLOP_open_document( )* geöffnet werden. Sie müssen sein Dokument-Handle an die Unteroption *visdoc* der Option *field* übergeben:

```
field={visdoc=<handle> rect={100 100 300 150}}
```

Position und Größe des Signaturfelds. Die Signaturoption *field* steuert die Darstellung der Signatur auf der Seite. Position und Größe der Visualisierungsseite für die Signatur auf der sichtbaren Seite des signierten Dokuments können mit der Option *rect* der Option *field* festgelegt werden. Die Größe kann explizit festgelegt werden oder implizit durch Angabe einer Ecke und ein oder zwei der anderen Abmessungen. Die fehlenden Werte werden mit dem Schlüsselwort *adapt* automatisch berechnet, um eine Verzerrung zu vermeiden. Mit dem Schlüsselwort *adapt* können Sie die Visualisierungsseite an eine beliebige Ecke des Signaturrechtecks anhängen. Das resultierende Rechteck darf nicht über die Seite hinausreichen. Die Beispiele unten zeigen verschiedene Kombinationen:

- Am einfachsten ist es, die Seite mit dem Signaturvisualisierung bereits in der gewünschten Größe für die Zielseite vorzubereiten. In diesem Fall können Sie einfach die Koordinaten der unteren linken Ecke des Feldes übergeben und PLOP DS wird die Original-Seitengröße für die Visualisierung der Signatur verwenden:

  ```
  rect={100 100 adapt adapt}
  ```

- An der linken unteren Ecke anhängen, die Breite erhalten und die Höhe anpassen, um eine Verzerrung zu vermeiden:

  ```
  rect={100 100 300 adapt}
  ```

- An der linken unteren Ecke anhängen, die Breite erhalten und die Höhe anpassen, um eine Verzerrung zu vermeiden:

  ```
  rect={100 100 adapt 200}
  ```

- Die Seite in das Rechteck einpassen, d.h. sowohl Breite als auch Höhe des Rechtecks bleiben erhalten. Falls die Seite und das Rechteck ein unterschiedliches Verhältnis von Breite und Höhe haben, wird die Signaturvisualisierung verzerrt dargestellt:

  ```
  rect={100 100 300 200}
  ```

Um ein geeignetes Signaturfeld-Rechteck abhängig von der Größe der Signaturvisualisierung zu berechnen, können Sie mit der pCOS-Schnittstelle die Seitengrößen abfragen (beachten Sie, dass die pCOS-Seitenindizes bei 0 beginnen):

```
width  = plop.pcos_get_number(visdoc, "pages[" + (vispage-1) + "]/width");
height = plop.pcos_get_number(visdoc, "pages[" + (vispage-1) + "]/height");
```

Signieren in einem bestehenden Formularfeld. Enthält das Eingabedokument bereits ein Signaturfeld, können Sie dieses für die Signatur und die Signaturvisualisierung verwenden. Dazu übergeben Sie den Namen des bestehenden Feldes, sofern bekannt:

```
field={name=MyExistingFieldName visdoc=<handle>}
```

Kennen Sie den Feldnamen nicht, können Sie PLOP instruieren, eine bestehendes Signaturfeld wie folgt zu verwenden:

```
field={fillexisting visdoc=<handle>}
```

Selbst wenn Sie in einem bestehenden Feld signieren, können Sie seine Position und Größe mit der Option *rect* ändern. Wenn Sie eine Signatur in einem bestehenden Feld erstellen und das Feld ein sichtbares Rechteck auf der Seite verwendet, müssen Sie die Option *visdoc* übergeben (oder das Feld mit der Option *rect={0 0 0 0}* unsichtbar machen).

Platzieren der Signaturvisualisierung im Signaturfeld. Die Seite mit der Signaturvisualisierung wird in das Signaturfeld platziert und so skaliert, dass sie unter Beibehaltung ihrer Proportionen vollständig in das Rechteck hineinpasst. Dies ist besonders nützlich, wenn Sie die Signatur in ein bestehendes Formularfeld platzieren möchten und die Proportionen von Breite und Höhe des Feldes nicht übereinstimmen.

Mit der Unteroption *position* der Option *field* lässt sich die Platzierung der Visualisierungsseite innerhalb des Signaturfelds festlegen.

Standardmäßig wird die Visualisierungsseite horizontal und vertikal im Feld zentriert. Dies lässt sich ändern, z.B. um die Visualisierungsseite an der unteren linken Ecke des Signaturfelds zu platzieren:

```
field={name=MyExistingFieldName visdoc=<handle> position={left bottom} }
```

pCOS. Die Sichtbarkeit von Signaturen wird in pCOS mit *signaturefields[...]/visible=true* ausgegeben. Ob ein Signaturfeld bereits eine Signatur enthält, können Sie mit *signaturefields[...]/sigtype != none* abfragen.

7.3.2 Konformität zu PDF/A, PDF/UA, PDF/X und PDF/VT

Soweit nicht anders in diesem Handbuch angegeben, sind alle PLOP-Operationen konform zu den Bestimmungen von PDF/A, PDF/UA, PDF/VT und PDF/X, die Standardkonformität wird von PLOP also erhalten. Es gibt jedoch einige Ausnahmen von dieser Regel, wenn PLOP-Operationen aufgrund einer bestimmten Norm unzulässig sind, z.B. Verschlüsselung in PDF/A. In diesen Fällen müssen Sie Ihre Prioritäten setzen:

- Muss die Standardkonformität eingehalten werden, werden die Operationen von PLOP abgewiesen. Dies ist das Standardverhalten.
- Ist eine Operation, z.B. Verschlüsselung, wichtiger als die Standardkonformität, können Sie den Standard-Identifikator mit der Option *sacrifice* entfernen.

Im Folgenden finden Sie detaillierte Hinweise zu den einzelnen Standards.

PDF/A-Konformität. Der PDF/A-Standard erlaubt CMS- und CAdES-basierte Signaturen. PDF/A-2 und PDF/A-3 empfehlen die Einbettung von Zeitstempel, Sperrinformationen und so viel von der Zertifikatskette wie verfügbar, dies ist jedoch nicht zwingend vorgeschrieben und wird daher von PLOP DS nicht erzwungen.

Im PDF/A-Modus, d.h., wenn die Eingabe PDF/A-konform ist und die Option *sacrifice* nicht auf *pdfa* gesetzt ist, muss ein Signaturbild bezüglich seiner PDF/A-Eigenschaften kompatibel sein:

- Die PDF/A-Konformitätsstufe des Visualisierungsdokuments muss kompatibel sein (siehe Tabelle 7.2).
- Die Druckausgabebedingung des Visualisierungsdokuments muss kompatibel sein (siehe Tabelle 7.3).

Tipp: ein Visualisierungsdokument vom Typ PDF/A-1a ohne Druckausgabebedingung (in Tabelle 7.2 und Tabelle 7.3 rot markiert) ist zu allen PDF/A-Teilen, Konformitätsstufen und Druckausgabebedingungen kompatibel. Das PLOP DS-Paket enthält eine Beispieldatei für eine Signaturvisualisierung mit diesen Eigenschaften (*signing_man_pdfa1a.pdf)*. Es kann zu Testzwecken für die Visualisierung von Signaturen für alle PDF/A-Varianten verwendet werden. Ein Visualisierungsdokument vom Typ PDF/A-1b ohne Druckausgabebedingung ist zur Konformitätsstufe *b* aller PDF/A-Teile kompatibel.

Wenn Sie keine PDF/A-Konformität benötigen, können Sie den Eintrag zur Standardkonformität mit folgender Option entfernen:

```
sacrifice={pdfa}
```

Tabelle 7.2 Kompatible PDF/A-Stufen des Visualisierungsdokuments für verschiedene PDF/A-Eingabestufen

	PDF/A-Stufe des Visualisierungsdokuments				
PDF/A-Stufe des Eingabedokuments	***PDF/A-1a:2005***	***PDF/A-1b:2005***	***PDF/A-2a, PDF/A-3a***	***PDF/A-2b, PDF/A-3b***	***PDF/A-2u, PDF/A-3u***
PDF/A-1a:2005	*zulässig*	–	–	–	–
PDF/A-1b:2005	*zulässig*	*zulässig*	–	–	–
PDF/A-2a, PDF/A-3a	*zulässig*	–	*zulässig*	–	–
PDF/A-2b, PDF/A-3b	*zulässig*	*zulässig*	*zulässig*	*zulässig*	*zulässig*
PDF/A-2u, PDF/A-3u	*zulässig*	–	*zulässig*	–	*zulässig*

Tabelle 7.3 Konformität der PDF/A-Druckausgabebedingung von Visualisierungsdokumenten (für alle PDF/A-Konformitätsstufen)

	Druckausgabebedingung des Visualisierungsdokuments			
Druckausgabebedingung des Eingabedokuments	***keine***	***Graustufen***	***RGB***	***CMYK***
keine	*zulässig*	–	–	–
ICC-Profil für Graustufen	*zulässig*	*zulässig*[1]	–	–
ICC-Profil für RGB	*zulässig*	–	*zulässig*[1]	–
ICC-Profil für CMYK	*zulässig*	–	–	*zulässig*[1]

1. Die Druckausgabebedingung des Visualisierungsdokuments und des Eingabedokuments müssen identisch sein.

PDF/UA-Konformität. Die PDF/UA-Anforderungen für unsichtbare Signatur-Formularfelder wurden gemäß des »Tagged PDF Best Practice Guide« gelockert (veröffentlicht 2017 vom *PDF/UA Competence Center* der *PDF Association).* Vor allem müssen unsichtbare Signaturfelder nicht mehr in die Strukturhierarchie aufgenommen werden und erfordern keine besonderen Vorbereitungen oder Signaturoptionen mehr.

Hinweis *Die Zugriffsprüfung von Acrobat DC unterstützt diese gelockerte Regel nicht. Für ein unsichtbares Signatur-Formularfeld, das nicht in der Strukturhierarchie enthalten ist, meldet sie nach wie vor »Tag-Anmerkungen - Fehlgeschlagen«.*

Um sichtbare Signaturfelder verwenden zu können, müssen Sie ein geeignetes Formularfeld mit Alternativtext im Eingabedokument vorbereiten; das Erstellen eines neuen Feldes ist nicht möglich. In Acrobat XI/DC kann dies für ein bestehendes PDF/UA-Dokument folgendermaßen erreicht werden:

- Anzeige in Acrobat DC: *Werkzeuge, Formulare vorbereiten…*, wählen Sie das Dokument aus, *Start*. Wählen Sie in der Liste der Formular-Werkzeuge in der Werkzeugleiste am oberen Rand das Signatur-Formularwerkzeug aus.
  Acrobat XI: Öffnen Sie das Werkzeugfenster und dort den Bereich *Formulare*. Wählen Sie *Erstellen*. Wählen Sie im nachfolgenden Dialog *Aus vorhandenem Dokument, Weiter, Aktuelles Dokument verwenden*. Wählen Sie unter *Formulare* den Eintrag *Aufgaben* und klicken Sie auf *Neues Feld hinzufügen, Digitale Unterschrift*.
- Platzieren Sie ein Formularfeld-Rechteck auf der Seite.
- Schließen Sie das Fenster *Formular vorbereiten* (Acrobat DC) oder klicken Sie auf *Formularbearbeitung schließen* (Acrobat XI) und öffnen Sie das Navigationsfenster *Tags*.
- Wählen Sie oben aus dem Optionsmenü des *Tags*-Fensters den Eintrag *Suchen…* .
- Wählen Sie im nachfolgenden Dialog *Nicht markierte Anmerkungen* und klicken Sie auf *Suchen*.
- Das zuvor erstellte Signaturfeld sollte nun markiert sein. Klicken Sie im Dialog *Element suchen* auf *Tag-Element* und wählen Sie *Typ: Formular*, optional können Sie den Feldtitel übergeben, und klicken Sie auf *OK*.
- Im Navigationsfensters *Tags* sollte nun das neue Strukturelement *Form* am Ende der Tagliste angezeigt werden. Wählen Sie dieses aus und ziehen Sie es an eine geeignete Position in der Tag-Hierarchie, entsprechend der Stelle im Strukturbaum, an der das Signaturfeld gelesen werden soll.
- Wir empfehlen, dem Signaturfeld einen Alternativtext zuzuweisen: Klicken Sie mit der rechten Maustaste auf das Strukturelement *Formular* in der Hierarchie, wählen Sie *Eigenschaften…* und geben Sie einen geeignete Alternativtext für das Feld ein.

Unter der Annahme, dass für das Signaturfeld der Name *Signature1* vergeben wurde, können Sie diesen Namen in der Signatur-Optionsliste als Zielfeld für die Signatur angeben:

```
field={name=Signature1}
```

Alternativ können Sie PLOP DS instruieren, die Signatur unabhängig von ihrem Namen in das vorhandene Feld zu platzieren:

```
field={fillexisting}
```

Mit der Unteroption *tooltip* der Signaturoption *field* können Sie einen geeigneten Alternativtext zum Signaturfeld für die Verwendung durch Screenreader-Software übergeben.

Wenn Sie keine PDF/UA-Konformität benötigen, können Sie den Eintrag zur Standardkonformität mit folgender Option entfernen:

```
sacrifice={pdfua}
```

Konformität zu PDF/X und PDF/VT. Visualisierung von Signaturen wird im PDF/X- und PDF/VT-Modus nicht unterstützt.

Wenn Sie keine PDF/X-Konformität benötigen, können Sie den Eintrag zur Standardkonformität mit folgender Option entfernen (ähnlich für PDF/VT):

```
sacrifice={pdfx}
```

7.3.3 Document Security Store (DSS)

Eine spezielle PDF-Datenstruktur namens *Document Security Store* (DSS) kann Zertifikate und zugehörige OCSP- und CRL-Informationen zur Zertifikatsperrung enthalten. Dieses Material wird zusammenfassend als Prüfinformationen bezeichnet und spielt eine wichtige Rolle bei der Langzeitvalidierung (LTV). Der DSS wurde mit PAdES Teil 4 eingeführt und in ISO 32000-2 aufgenommen. Er wird ab Acrobat X unterstützt. Während der DSS für Genehmigungs- und Zertifizierungssignaturen optional ist, ist er für die LTV-Fähigkeit von Zeitstempelsignaturen auf Dokumentebene und Signaturen mit eingebettetem Zeitstempel zwingend erforderlich.

Speichern von Prüfinformationen im DSS statt im Signaturobjekt reduziert die Dateigröße, denn anders als das Signaturobjekt kann der DSS komprimiert werden und erfordert keine ASCII-Darstellung (die die Größe des Signatur verdoppelt). Außerdem kann der DSS Daten für die Validierung von mehreren Signaturen aufnehmen, während das Signaturobjekt nur Prüfinformationen für eine einzelne Signatur umfasst.

Einige Prüfinformationen können nur im Signaturobjekt gespeichert werden, einige nur im DSS und manche an beiden Stellen. Mit der Signaturoption *dss* können Sie den Speicherort der Elemente in der letzten Gruppe steuern. In Tabelle 7.4 werden beide Speicherorte miteinander verglichen.

Tabelle 7.4 Speicherorte für Prüfinformationen

	Signaturobjekt	***Document Security Store (DSS)***	***gesteuert von Option dss***
Signaturzertifikat	*ja*	*–*	*–*
TSA-Zertifikat	*–*	*ja*	*–*
Zertifikate außer Signatur- und TSA-Zertifikat (z.B. Aussteller des Signaturzertifikats) sowie zugehörige OCSP-Antworten und CRLs	*ja*	*ja*	*ja*
OCSP-Antworten und CRL für das Signaturzertifikat	*ja*	*ja*	*ja*
OCSP-Antworten und CRLs für TSA-Zertifikate[1]	*–*	*ja*	*–*

1. Wenn Prüfinformationen für Zeitstempel eingebettet werden müssen, hängt PLOP DS immer einen DSS als inkrementelles PDF-Update an.

PLOP DS bewahrt einen vorhandenen DSS mit Prüfinformationen für frühere Signaturen, die im Eingabedokument enthalten sind. Der neue DSS enthält die Inhalte des bestehenden DSS sowie Prüfinformationen für die neue Signatur. Dadurch wird sichergestellt, dass der LTV-Status einer vorhandenen Signatur erhalten bleibt.

In Acrobat X und höher können Sie einen DSS zu einem signierten Dokument hinzufügen, indem Sie das Navigationsfenster *Unterschriften* öffnen und im Optionsmenü *Prüfinformationen hinzufügen* auswählen.

pCOS. Das Vorhandensein eines DSS lässt sich mit dem pCOS-Pfad *type:/Root/DSS* abfragen, der den Wert 6 *(dict)* hat, wenn ein DSS vorhanden ist. Beachten Sie, dass ein DSS an sich nicht automatisch den LTV-Status garantiert, da er nur eine Teilmenge der erforderlichen Zertifikate und Sperrinformationen enthalten könnte.

7.3.4 Signaturen und inkrementelle PDF-Updates

Standardmäßig hängt PLOP DS digitale Signaturen an das Eingabedokument an, und zwar mit Hilfe eines sogenannten inkrementellen PDF-Updates: Zunächst wird eine Kopie des Eingabedokuments erstellt. Dann werden die Signaturdaten ans Ende angehängt, wodurch Inhalt und Struktur des Originaldokuments erhalten bleiben. Mit der Signaturoption *update=false* schreibt PLOP DS die Hierarchie der PDF-Objekte neu *(Rewrite),* anstatt ein inkrementelles PDF-Update hinzuzufügen. Tabelle 7.5 vergleicht Signaturen im Update- und Rewrite-Modus.

Tabelle 7.5 Signieren im Update- oder Rewrite-Modus

	Update-Modus `(update=true)`	***Rewrite-Modus*** `(update=false)`
vorhandene Signaturen	*bleiben erhalten*	*gehen verloren*[1]
DSS für Genehmigungs- und Zertifizierungssignaturen kann hinzugefügt werden	*ja*	*ja*
DSS für Zeitstempelsignaturen auf Dokumentebene und Signaturen mit eingebettetem Zeitstempel kann hinzugefügt werden	*ja*	*–*[2]
vorhandener DSS bleibt erhalten	*ja*	*ja*
Verschlüsselung mit neuen Parametern möglich (`userpassword`, `masterpassword`, `permissions`)	*–*	*ja*
Optimierung möglich	*–*	*ja*
Reparaturmodus für beschädigte Eingabedokumente möglich	*–*	*ja*
Signaturgeschwindigkeit	*etwas schneller*	*etwas langsamer*
vorige Dokumentversion (vor Anwendung der Signatur) kann in Acrobat wiederhergestellt werden	*ja*	*nein*

1. Das Signieren von Dokumenten mit vorhandenen Signaturen im Rewrite-Modus erfordert `sacrifice={signatures}`*, wodurch die Signaturen entfernt werden. Wird die Option* `sacrifice` *nicht angegeben, werden signierte Eingabe-Dokumente abgelehnt.*
2. Wenn Prüfinformationen für Zeitstempel eingebettet werden, hängt PLOP DS immer einen DSS als inkrementelles PDF-Update an.

Signieren von beschädigten Dokumenten. Wenn im Update-Modus signiert wird, können Fehler in den PDF-Querverweistabellen oder der Objektstruktur des Dokuments nicht repariert werden. Wenn ein Dokument bei *PLOP_open_document( )* repariert wer-

den muss und anschließend im Update-Modus signiert wird, bricht *PLOP_create_document()* mit folgender Fehlermeldung ab:

```
Cannot sign damaged input document 'bad.pdf' in update mode; use update=false
(invalid xref table)
```

Um beschädigte Dokumente bereits bei *PLOP_open_document()* zu identifizieren, können Sie die Option *repair=none* übergeben. Als Ergebnis schlägt *PLOP_open_document()* für beschädigte Dokumente fehl. Um zu reparierende Dokumente zu signieren, verwenden Sie die Option *update=false*; für weitere Informationen siehe Tabelle 7.5.

Wiederherstellen früherer Revisionen eines signierten Dokuments. Da inkrementelle Updates nur Informationen zu einem Dokument hinzufügen, bleibt die Struktur des Eingabedokuments erhalten. Wenn ein signiertes Dokument geändert wird, kann die signierte Revision rekonstruiert werden, indem die inkrementellen Updates entfernt werden. In Acrobat XI/DC kann dies folgendermaßen erreicht werden:

- Öffnen Sie das Unterschriftsfenster, wählen Sie eine Signatur aus und klicken Sie auf das Plus-Zeichen.
- Wählen Sie *Klicken Sie, um diese Version anzuzeigen,* um die signierte Revision wiederherzustellen.

Ist die Signatur über einen DSS in einem separaten inkrementellen Update LTV-fähig gemacht worden, wird dieses Update entfernt, wenn die signierte Revision wiederhergestellt wird. Als Ergebnis kann die Signatur in der vorherigen Revision möglicherweise nicht mehr als LTV-fähig angezeigt werden, obwohl die selbe Signatur im vollständigen Dokument noch als LTV-fähig angezeigt wurde. Dies wird durch das Entfernen von inkrementellen PDF-Updates ausgelöst; der tatsächliche LTV-Status der Signaturen im gesamten Dokument ist davon nicht betroffen. Dieses Problem gilt nicht für Signaturen mit eingebettetem Zeitstempel, da Acrobat für die TSA keine vollständigen Prüfinformationen benötigt.

Dieser Effekt tritt nur auf, wenn die Prüfinformationen in einem DSS in einem inkrementellen Update angehängt werden, und kann daher auf zwei Arten vermieden werden:

- setzen Sie *dss=false,* um den DSS zu vermeiden;
- setzen Sie *update=false,* um inkrementelle Updates zu vermeiden.

Beide Optionen haben keine Auswirkungen auf Dokument-Zeitstempel und eingebettete Zeitstempel, die immer einen DSS in einem inkrementellen Update erfordern.

pCOS. Die Anzahl der Dokument-Revisionen bei inkrementellen Updates wird in pCOS mit dem Pseudo-Objekt *revisions* ausgegeben. Während jede Signatur eine neue Revision erzeugt, können auch durch andere Änderungen, wie z.B. das Hinzufügen eines DSS, neue Revisionen entstehen. Die Anzahl der Revisionen kann daher größer sein als die Anzahl der Signaturen im Dokument.

7.3.5 Kombination von Verschlüsselung und Signaturen

Die Kombination von Verschlüsselung und Signaturen erfordert besondere Aufmerksamkeit, da eine Signatur ungültig würde, wenn ein Dokument zuerst signiert und dann verschlüsselt wird. Sie können entweder in einem einzigen Durchgang verschlüs-

seln und signieren oder ein verschlüsseltes Eingabedokument im Update-Modus signieren.

Verschlüsseln und Signieren in einem Durchgang. Der einfachste Ansatz ist, das Dokument in einem einzigen Durchgang zu verschlüsseln und zu signieren. Da die Eingabedatei geändert werden muss, ist das Signieren nur im Rewrite-Modus möglich. Verschlüsselungsoptionen, d.h. *userpassword* oder *masterpassword* oder Empfängerzertifikate können zusammen mit Signaturoptionen übergeben werden. Wird eine dieser Verschlüsselungseinstellungen übergeben, wird die Signatur im Rewrite-Modus angewendet, d.h. *update* wird auf *false* gesetzt.

Signieren verschlüsselter Eingabedokumente im Update-Modus. Verschlüsselte Eingabedokumente können im Update-Modus signiert werden. Allerdings können die Verschlüsselungseinstellungen in diesem Fall nicht geändert werden. Dies hat folgende Konsequenzen:

- Das Master-Kennwort des Eingabedokuments muss mit der Option *password* übergeben werden (oder bei Dokumenten mit Zertifikatsicherheit eine passende digitale ID).
- Wird *update=true* übergeben, sind die Optionen *encryption*, *masterpassword*, *permissions* und *userpassword* unzulässig und *PLOP_add_recipient()* darf nicht aufgerufen werden, da die Werte des Eingabedokuments für das Ausgabedokument verwendet werden.

7.3.6 Zertifizierungssignaturen

Für eine Einführung zu Zertifizierungs- (Autoren)signaturen siehe »Zertifizierungssignaturen«, Seite 92. Beim Öffnen eines Dokuments mit einer Zertifizierungssignatur wird in Acrobat am oberen Rand in der blauen Dokumentmitteilungsleiste eine Schleife angezeigt sowie im Navigationsfenster *Unterschriften* von Acrobat (wieder mit einer Schleife für eine gültige Signatur). Zertifizierungssignaturen legen fest, welche Art von Änderungen an einem Dokument vorgenommen werden dürfen, ohne die Signatur ungültig zu machen (siehe Abbildung 7.6 und Tabelle 7.6). Zertifizierungssignaturen können in PLOP DS mit der Signaturoption *certification* erstellt werden.

Mit folgenden Signaturoptionen lässt sich eine Zertifizierungssignatur so erstellen, dass das Ausfüllen von Formularfeldern erlaubt ist, ohne die Signatur ungültig zu machen:

```
digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ↵
certification=formfilling
```

Mit der Unteroption *preventchanges* lassen sich die Werkzeuge in der Benutzeroberfläche von Acrobat deaktivieren, die die Signatur ungültig machen würden, wie z.B. die Kommentarwerkzeuge. Auf diese Weise kommen Benutzer nicht in Versuchung, Änderungen durchzuführen, die die Zertifizierungssignatur ungültig machen würden. Bei der Zertifizierung von Dokumenten in Acrobat werden Änderungen immer verhindert. Die Option *preventchanges* ist standardmäßig auf *true* gesetzt. Bei *preventchanges=false* werden in Acrobat alle Werkzeuge aktiviert. Unzulässige Änderungen machen die Zertifizierungssignatur dennoch ungültig.

Da eine Zertifizierungssignatur immer die erste Unterschrift in einem Dokument sein muss, sollte sie nicht auf ein Dokument angewendet werden, das bereits eine Signatur enthält.

Tabelle 7.6 Zulässige Dokumentänderungen, die die Zertifizierungssignatur nicht ungültig machen

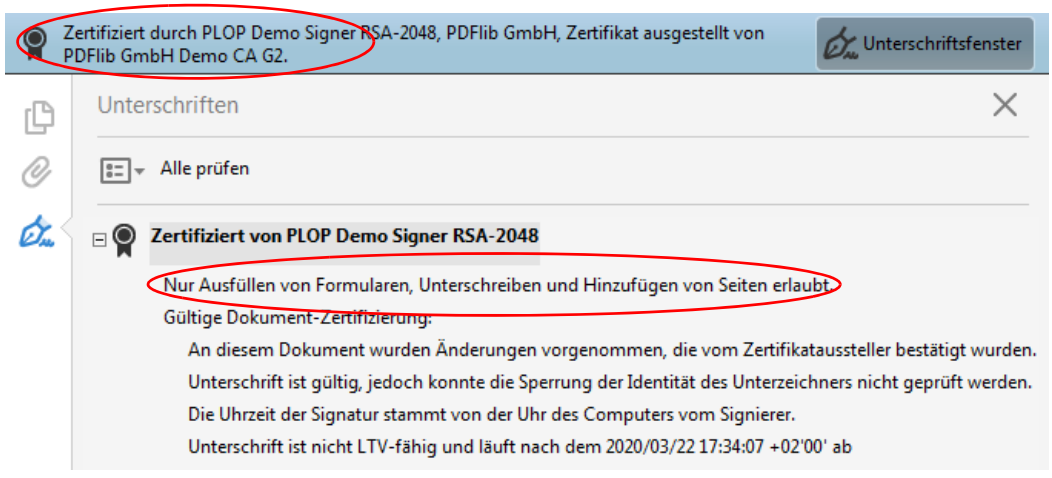
Signaturtyp (Optionsliste)	**Zulässige Änderungen, die die Zertifizierungssignatur erhalten**				
	Formularfelder ausfüllen	*digitales Signieren und Hinzufügen von Seiten*[1]	*Erzeugen, Löschen oder Ändern von Anmerkungen*	*Hinzufügen von Signaturfeldern*[2]	*alle anderen Änderungen*
`certification=nochanges`	–	–	–	–	–
`certification=formfilling`	*ja*	*ja*[3]	–	–	–
`certification=formsandannotations`	*ja*	*ja*[3]	*ja*	–	–
`certification=none` *(d.h. Genehmigungssignatur oder Zeitstempelsignatur auf Dokumentebene)*	*ja*	*ja*	*ja*	*ja*	–

1. Hinzufügen von Seiten durch Kopieren eines Seiten-Templates ist zulässig (diese Methode wird jedoch selten verwendet), nicht jedoch das manuelle Hinzufügen von Seiten mit `Werkzeuge, Seiten, Seiten einfügen`.
2. Hinzufügen von Signaturfeldern über `Füllen und unterschreiben, Unterschrift platzieren` *ist zulässig, nicht jedoch das Hinzufügen von Formularfeldern mit* `Werkzeuge, Formulare, Bearbeiten`.
3. Nur Signieren durch Klicken auf ein Signaturfeld ist zulässig, nicht jedoch über das Acrobat-Menü.

Gültigkeit von Zertifizierungssignaturen in Acrobat. Selbst wenn eine Zertifizierungssignatur technisch gültig ist, gibt es einige zusätzliche Anforderungen, um die Vorteile von zertifizierten Dokumenten in Acrobat vollständig zu nutzen:

- Zertifizierungssignaturen lassen sich am einfachsten mit Zertifikaten einer AATL-Zertifizierungsstelle erstellen (siehe »Vertrauenswürdige Stammzertifikate in Acrobat (Trusted Root Certificates)«, Seite 93). Da für die Adobe-Stammzertifizierungs-

Abb. 7.6 Zertifizierungssignatur mit »Ausfüllen von Formularen und Unterschreiben erlaubt« in Acrobat



stelle automatisch die erforderlichen Einstellungen für die Vertrauenswürdigkeit gesetzt sind, ist keine weitere Konfiguration notwendig.

- Wenn Endbenutzer-Zertifikate unter einer Stammzertifizierungsstelle, die Acrobat nicht bekannt ist, zum Erstellen von Zertifizierungssignaturen verwendet werden sollen, empfiehlt es sich, dem Stammzertifikat in Acrobat die notwendige Vertrauensstufe wie folgt zuzuweisen:
  Bearbeiten, Voreinstellungen..., Unterschriften, Identitäten und vertrauenswürdige Zertifikate, Weitere..., Vertrauenswürdige Zertifikate, wählen Sie das Stammzertifikat, *Zertifikatberechtigung bearbeiten* und aktivieren Sie *Zertifizierte Dokumente*.
  Als Ergebnis werden alle Zertifizierungssignaturen als gültig erachtet, die mit diesem Zertifikat unter dem ausgewählten Stamm erstellt wurden.
- Auch für ein einzelnes Zertifikat können Sie die erforderliche Vertrauensstufe zuweisen. Dies ist jedoch eher ungewöhnlich und wird nicht empfohlen. Gehen Sie folgendermaßen vor:
  Öffnen Sie das Navigationsfenster *Unterschriften*, wählen Sie für die Zertifizierungssignatur die *Zertifikatdetails...,* wählen Sie das Signaturzertifikat aus der Zertifikatskette (d.h. das unterste in der Liste), öffnen Sie die Registermarke *Vertrauenswürdigkeit*, klicken Sie auf *Zu vertrauenswürdigen Zertifikaten hinzufügen...*, bestätigen Sie die nachfolgende Informationsmeldung mit *OK* und bearbeiten Sie die Einstellungen zur Vertrauenswürdigkeit.

Wenn Sie keine der oben beschriebenen Vorgehensweisen anwenden, markiert Acrobat die Zertifizierungssignatur mit einem gelben Dreieck statt der Schleife und fügt folgenden Text hinzu: »Das Zertifikat des Unterzeichners wurde zur Erstellung zertifizierter Dokumente als nicht vertrauenswürdig eingestuft«.

pCOS. Zertifizierungssignaturen werden in pCOS als *signaturefields[...]/sigtype=certification* ausgegeben. Welche Art von Änderungen erlaubt ist, kann mit *signaturefields[...]/permissions* abgefragt werden, das eins der Schlüsselwörter *nochanges, formfilling* oder *formsandannotations* zurückgibt.

Mit dem Pseudo-Objekt *signaturefields[...]/preventchanges* lässt sich prüfen, ob Elemente auf der Acrobat-Benutzeroberfläche deaktiviert werden, damit die Zertifizierungssignatur nicht aus Versehen durch unzulässige Änderungen ungültig gemacht wird.

Signieren zertifizierter Dokumente, bei denen Änderungen verboten sind. Da eine Zertifizierungssignatur die erste Signatur in einem Dokument sein muss, können in der Regel nach der ersten Signatur weitere Signaturen zugewiesen werden. Eventuell wurde die Zertifizierungssignatur jedoch mit der Einstellung »Keine Änderungen zulässig« (in PLOP DS: *certification=nochanges)* erstellt. Diese Art von Eingabedokument führt zu einem Konflikt, der durch den Anwendungsentwickler gelöst werden muss: Während die Datei alle Änderungen einschließlich Signaturen verbietet, will die Anwendung signieren, wodurch die Zertifizierungssignatur ungültig würde (im Gegensatz zu Genehmigungssignaturen, die zusätzliche Signaturen im Update-Modus zulassen, ohne ungültig zu werden). Dies gilt auch für das Signieren im Update-Modus, da die Zertifizierung keinerlei Änderungen erlaubt. Der Konflikt kann auf folgende Arten gelöst werden:

- Standardverhalten: Die Eingabe wird zurückgewiesen, da die vorhandene Zertifizierungssignatur Priorität gegenüber der neuen Signatur hat.

- Die Zertifizierungssignatur wird entfernt und die neue Signatur angewendet. Verwenden Sie hierzu die Option *sacrifice=signatures* von *PLOP_create_document( )*.

7.4 Sperrinformationen zu Zertifikaten

Eine Signatur kann optional Informationen über den Status der Zertifikatsperrung des Signaturzertifikats enthalten. Mit diesen Sperrinformationen kann die Validierungssoftware prüfen, ob das Zertifikat zum Zeitpunkt der Unterzeichnung noch gültig (nicht gesperrt) war. Dies lässt sich auf zwei Arten erreichen.

In Acrobat XI/DC können Sie die Sperrinformationen folgendermaßen in der Zertifikatanzeige anzeigen lassen: öffnen Sie das Navigationsfenster *Unterschriften*, rechtsklicken Sie auf die Signatur und wählen Sie *Unterschriftseigenschaften einblenden..., Zertifikat anzeigen...* und wählen Sie die Registerkarte *Sperrung* (siehe Abbildung 7.7 und Abbildung 7.8).

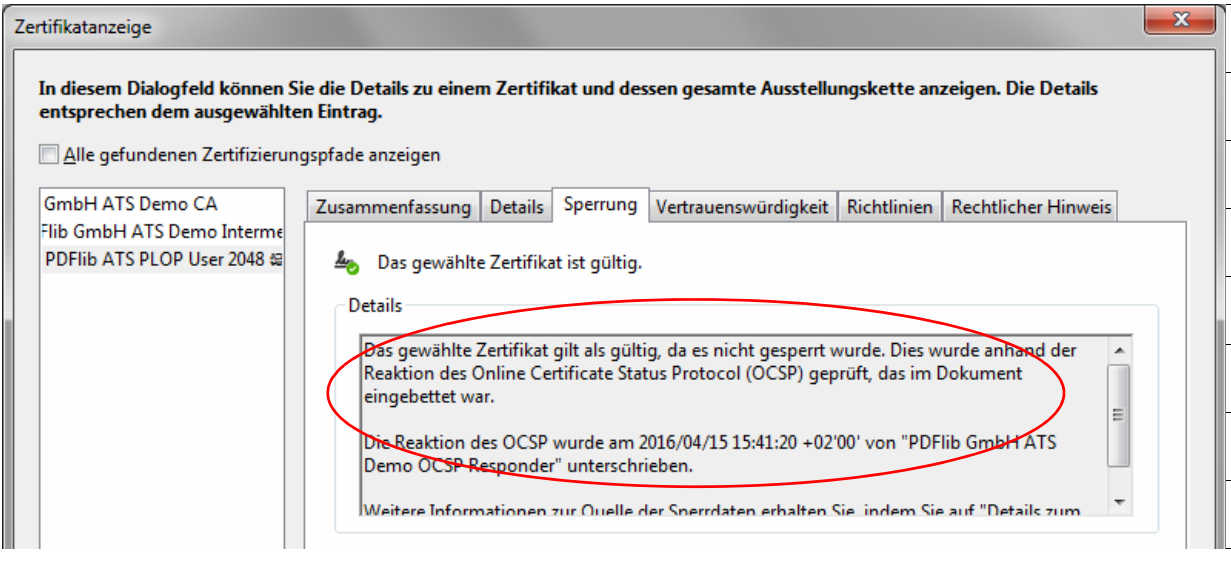
7.4.1 Online Certificate Status Protocol (OCSP)

Hinweis *Die Einbettung von OCSP-Antworten wird für* `engine=mscapi` *nicht unterstützt.*

Überblick über OCSP. Bei Verwendung von OCSP gemäß RFC 2560 und RFC 6960 sendet die Signatursoftware eine Netzanfrage an den OCSP-Server (auch OCSP-Responder genannt), um den Zertifikatstatus in Echtzeit abzufragen. Ein OCSP-Responder ist ein Server mit Echtzeitzugang zur CA-Datenbank mit den veröffentlichten und gesperrten Zertifikaten. Der OCSP-Responder prüft, ob das Zertifikat zum Zeitpunkt der Anfrage gültig ist und gibt eine signierte Antwort (`Response`) mit dem Ergebnis zurück. Diese OCSP-Antwort wird in die Signatur eingebettet.

Ein Zertifikat kann eine Erweiterung namens *Authority Info Access, AIA* mit der Zugriffsmethode *ocsp* gemäß RFC 3280 enthalten. Dies ist meist für AATL-Zertifikate der Fall (siehe »Adobe Approved Trust List (AATL)«, Seite 93). Diese Erweiterung enthält eine URL für einen OCSP-Responder, der mit der CA assoziiert ist, die das Zertifikat ausgestellt hat. Alternativ kann die URL mit der Signaturoption *ocsp* übergeben werden. Wenn PLOP DS eine OCSP-Anfrage für ein bestimmtes Zertifikat schickt, gibt der OCSP-

Abb. 7.7
Anzeige von OCSP-Informationen in Acrobat



Server eine signierte Antwort mit dem Status *good*, *revoked* oder *unknown* für das Zertifikat zurück. Um eine OCSP-Antwort mit dem Status *good* zu erzeugen, müssen folgende Bedingungen erfüllt sein:

- In der digitalen ID muss eine AIA-Erweiterung mit der Zugriffsmethode *ocsp* vorhanden sein oder die Unteroption *source* der Signaturoption *ocsp* muss übergeben werden.
- Der OCSP-Responder kann im Netzwerk über die angegebene URL erreicht werden und sendet eine Antwort innerhalb der Zeitspanne, die in der Unteroption *timeout* der Unteroption *source* der Signaturoption *ocsp* angegeben ist.
- Die OCSP-Antwort enthält den Status *good*, d.h. das Zertifikat, das von der Zertifizierungsstelle (CA) ausgegeben wurde, die durch den OCSP-Responder versorgt wird, muss gültig sein (also das Ende seiner Gültigkeitsdauer noch nicht erreicht haben) und darf nicht gesperrt sein.
- Der Signaturzeitpunkt muss in dem Zeitraum zwischen *thisUpdate* und *nextUpdate* in der OCSP-Antwort liegen oder (falls *nextUpdate* nicht vorhanden ist) darf der Signaturzeitpunkt nicht nach *thisUpdate* plus dem in der Unteroption *freshness* angegebenen Wert liegen. Beide Prüfungen erlauben eine Toleranz bis zu dem in der Unteroption *maxclockskew* angegebenen Wert, um Netzwerkverzögerungen oder ungenaue Systemzeit zu kompensieren.

Enthält die OCSP-Antwort den Status *good*, wird sie von PLOP DS in die erzeugte Signatur eingebettet. Andernfalls wird die unbrauchbare Antwort entweder ignoriert oder, abhängig von der Unteroption *critical,* keine Signatur erstellt. Standardmäßig verwendet PLOP DS die URL des OCSP-Responders in der AIA-Erweiterung, falls sie in der digitalen ID des Unterzeichners vorhanden ist, und ignoriert OCSP-Antworten ohne den Status *good*. Wenn die Einbettung von OCSP-Antworten jedoch mit der Option *ocsp* explizit angefragt wird, wird eine Antwort vom Typ *good* für die Erstellung der Signatur benötigt, sofern die Option *critical* nicht auf *false* gesetzt wurde.

OCSP-Konfiguration. Abhängig von der verwendeten PKI sollten Sie bei OCSP-Antworten folgende Konfigurationsaspekte beachten:

- Enthält das Zertifikat keine AIA-Erweiterung, müssen Sie mit der Unteroption *source* der Option *ocsp* einen OCSP-Responder übergeben.
- Zur Erstellung der OCSP-Anfrage wird ein gültiges Zertifikat für den Aussteller des Signaturzertifikats benötigt. Dies ist oft in der digitalen ID des Unterzeichners vorhanden; andernfalls muss es separat mit einer der Signaturoptionen *rootcertdir/ rootcertfile/certfile* übergeben werden.
- Da der OCSP-Responder eventuell eine Authentisierung für eine erfolgreiche Netzwerkkommunikation verlangt, werden für OCSP-Anfragen mehrere Authentisierungsoptionen unterstützt.
- Die OCSP-Funktion *nonce* vereitelt zwar Replay-Angriffe, verhindert aber auch Caching und reduziert daher die Leistung. Abhängig von der Konfiguration des OCSP-Responders müssen Sie eventuell die Option *nonce* übergeben. Bei einer Meldung mit etwa folgendem Inhalt wird die Funktion *nonce* vom OCSP-Responder nicht unterstützt. In diesem Fall können Sie diese Funktion mit der Option *nonce=false* abschalten:

```
OCSP response from URL 'http://ocsp.acme.com' for certificate 'CN = PDFlib GmbH...'
does not contain nonce although it was requested
```

Der OCSP-Responder von Microsoft weist die Anforderung mit der Fehlermeldung *unauthorized* zurück, falls er nicht für die *nonce*-Verarbeitung konfiguriert ist, aber ein *nonce* angefordert wird. In diesem Fall müssen Sie auch die Signaturoption *nonce=false* übergeben, um die *nonce*-Funktion zu deaktivieren.

- Mit der Unteroption *hash* der Option *ocsp* lässt sich eine Hash-Funktion auswählen, mit der in der OCSP-Anfrage und -Antwort Zertifikate identifiziert werden. Acrobat XI/DC kann für die Validierung von Signaturen jedoch nur OCSP-Antworten verarbeiten, die die Hash-Funktion SHA-1 verwenden. Aus diesem Grund erfordern Werte ungleich *sha1* die Signaturoption *conformance=extended*.

Sperrungsprüfung für das Zertifikat des OCSP-Responder. Das Signaturzertifikat des OCSP-Responders muss zum Zeitpunkt der Erzeugung der OCSP-Antwort gültig sein. Um rekursive Probleme (das Zertifikat des OCSP-Responders benötigt eine weitere OCSP-Antwort) zu vermeiden, sollte das Zertifikat des OCSP-Responders die Erweiterung *id-pkix-ocsp-nocheck* gemäß RFC 2560 enthalten. Dies ist bei den meisten kommerziellen OCSP-Respondern der Fall. Alternativ kann dieses Zertifikat auch die Erweiterung *CRL distribution points (CRLdp)* enthalten.

Beispiele für OCSP-Optionslisten. In den folgenden Beispielen wird nur der Teil der Optionsliste aufgeführt, der für die Einbettung der OCSP-Antwort relevant ist. Je nach Bedarf müssen Sie weitere Signaturoptionen hinzufügen.

Einbettung der OCSP-Antwort mit der in der digitalen ID des Unterzeichners vorhandenen URL und Abbruch mit einer Fehlermeldung, wenn in der digitalen ID keine AIA-Erweiterung mit der Zugriffsmethode *ocsp* verfügbar ist:

```
ocsp={source={}}
```

oder äquivalent

```
ocsp={}
```

Anfordern einer OCSP-Antwort unter Verwendung der AIA-Erweiterung, sofern vorhanden, und andernfalls Ignorieren aller Fehler:

```
ocsp={source={} critical=false}
```

oder äquivalent

```
ocsp={critical=false}
```

Keine Einbettung einer OCSP-Antwort, selbst wenn die AIA-Erweiterung in der digitalen ID vorhanden ist:

```
ocsp=none
```

Explizite Übergabe der URL und eines Zeitlimits von einer Sekunde für den OCSP-Responder und damit Überschreiben eines Eintrags in der AIA-Erweiterung, die in der digitalen ID eventuell vorhanden ist:

```
ocsp={source={url={http://ocsp.acme.com/} timeout=1000} }
```

Sicherstellen, dass erfolglose OCSP-Versuche das Signieren verhindern, und Deaktivieren der Funktion *nonce* für OCSP-Responder, die diese nicht unterstützen:

```
ocsp={critical nonce=false}
```

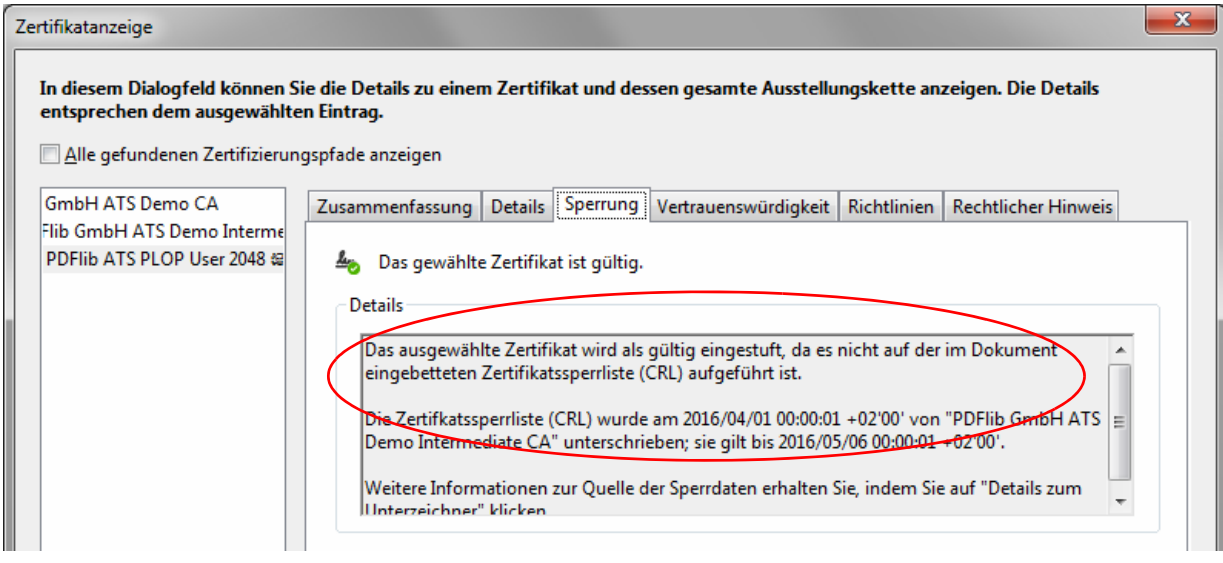
7.4.2 Zertifikatsperrlisten (Certificate Revocation Lists)

Hinweis *Die Einbettung von Certificate Revocation Lists (CRLs) wird für* `engine=mscapi` *nicht unterstützt.*

Überblick über Zertifikatsperrlisten (CRLs). Werden CRLs gemäß RFC 3280 verwendet, erstellt die Zertifizierungsstelle (CA) in regelmäßigen Abständen, z.B. einmal pro Tag, eine signierte Liste von Zertifikaten, die noch nicht abgelaufen, aber gesperrt sind. Diese wird der Signatursoftware zur Verfügung gestellt und in die Signatur eingebettet. Die Liste kann über das Netzwerk abgerufen bzw. lokal gespeichert werden. CRLs haben eine begrenzte Lebensdauer (z.B. einen Tag) und müssen vor dem Ende ihrer Lebensdauer erneuert werden. Da sie eine beliebige Menge von gesperrten Zertifikaten enthalten können, sind sie generell größer als OCSP-Antworten (bis zu mehreren Megabytes), wobei ihre Größe im Voraus nicht bekannt ist. Da in der PDF-Ausgabe immer die vollständige CRL eingebettet wird, werden die signierten PDF-Dokumente dadurch entsprechend aufgebläht. PLOP DS kann CRLs aus verschiedenen Quellen beziehen:

- Ein Zertifikat kann eine Erweiterung namens *CRL distribution points (CRLdp)* enthalten. Dies ist für AATL-Zertifikate immer der Fall (siehe »Adobe Approved Trust List (AATL)«, Seite 93). Die Erweiterung enthält eine oder mehrere Netzwerk-URLs von CRL-Ressourcen. PLOP DS durchsucht alle Einträge in der Erweiterung *CRLdp*, bis es eine CRL abrufen kann. Wurde eine verwendbare CRL gefunden, wird sie in die Signatur oder einen *Document Security Store* (DSS) eingebettet (siehe Abschnitt 7.3.3, »Document Security Store (DSS)«, Seite 110). Abhängig von der Verfügbarkeit einer OCSP-Antwort und den entsprechenden *critical*-Optionen wird die Erweiterung *CRLdp* für jedes Zertifikat ausgewertet, für das eine CRL benötigt wird.
- Alternativ zur Erweiterung *CRLdp* kann die Abfrage einer CRL für das Signaturzertifikat mit der Option *crl* erfolgen. Die Unteroption *source* zeigt auf eine Netzwerk-

Abb. 7.8
Anzeige von CRL-Informationen in Acrobat



Adresse für die dynamische Abfrage einer CRL; die Unteroption *filename* zeigt auf eine statische lokale CRL-Datei in DER-Kodierung.

- Eine oder mehrere lokale CRL-Dateien für das Signaturzertifikat und alle anderen beteiligten Zertifikate können in PEM-Kodierung mit den Signaturoptionen *crldir/crlfile* übergeben werden.

Ist das Zertifikat des Unterzeichners in der CRL enthalten, wurde es von der ausstellenden Zertifizierungsstelle (CA) gesperrt, d.h. es kann zur Erstellung einer gültigen Signatur nicht länger verwendet werden. In diesem Fall scheitert der Funktionsaufruf *PLOP_prepare_signature()* mit folgender Fehlermeldung:

```
Certificate verification failure for certificate with subject 'C = DE, L = Munich, O =
PDFlib GmbH, CN = PLOP Demo Signer RSA-2048': certificate revoked
```

PLOP DS verwendet CRLs so lange, bis sie ihre Gültigkeit verlieren. Nur wenn die Lebensdauer einer bestimmten CRL beendet ist und sie daher nicht mehr verwendet werden kann, lädt PLOP DS eine neue CRL vom Server herunter.

Beispiele für CRL-Optionslisten. In den folgenden Beispielen wird nur der Teil der Optionsliste aufgeführt, der für die Einbettung der CRL relevant ist. Je nach Bedarf müssen Sie weitere Signaturoptionen hinzufügen.

Einbettung der CRL mit der in der digitalen ID des Unterzeichners und Abbruch mit einer Fehlermeldung, wenn die Erweiterung *CRLdp* in der digitalen ID nicht verfügbar ist:

```
crl={source={}}
```

oder äquivalent

```
crl={}
```

Anfordern einer CRL unter Verwendung der Erweiterung *CRLdp*, wenn möglich, aber Ignorieren aller Fehler:

```
crl={source={} critical=false}
```

oder äquivalent

```
crl={critical=false}
```

Kein Abfordern einer CRL für das Signaturzertifikat oder für ein anderes Zertifikat, selbst wenn die Erweiterung *CRLdp* in der digitalen ID vorhanden ist. Dies ist sinnvoll, wenn die Online-Abfrage ohnehin scheitern würde, da zum Beispiel der Server offline ist:

```
crl=none
```

Explizite Übergabe der URL und eines Zeitlimits von einer Sekunde für den CRL-Server und damit Überschreiben eines Eintrags in der Erweiterung *CRLdp*, die in der digitalen ID eventuell vorhanden ist:

```
crl={source={url={http://crl.acme.com/} timeout=1000} }
```

Übergabe einer CRL, die in einer lokalen Datei auf der Festplatte enthalten ist:

```
crlfile={certs.pem}
```

7.4.3 OCSP oder CRL?

Folgende Faktoren sind relevant, wenn Sie die am besten geeignete Methode für die Übergabe von Sperrinformationen auswählen wollen:

- OCSP bietet Informationen zum Zertifikatstatus in Echtzeit. Da eine OCSP-Antwort jeweils nur ein einzelnes Zertifikat beschreibt, hat sie eine vorhersehbare Größe von nur wenigen Kilobytes. Auf der anderen Seite benötigt OCSP immer eine Netzverbindung zum OCSP-Responder.
- Der Vorteil von CRLs gegenüber OCSP ist, dass sie lokal gespeichert und somit die Netzlast gering gehalten werden kann. Der Nachteil ist, dass eine lokal gespeicherte CRL veralten kann, wenn sie nicht regelmäßig erneuert wird (d.h. veröffentlicht und heruntergeladen wird).
- Da CA-Zertifikate nur selten gesperrt werden müssen, sind CRLs für CAs in der Regel viel kleiner als CRLs für Endnutzer-Zertifikate.
- Da HSMs selten aufgebrochen oder gestohlen werden, sind CRLs für ein HSM-basiertes Zertifikat in der Regel auch sehr klein (vorausgesetzt, die CRL umfasst nur HSM-basierte Zertifikate und keine dateibasierten Zertifikate).
- Einige Gesetzgebungen oder private Signaturrichtlinien können eventuell eine der beiden Methoden erfordern oder verbieten.

Standardmäßig bettet PLOP DS eine CRL nur dann in die Signatur ein, wenn keine gültige gute OCSP-Antwort verfügbar ist. Dies lässt sich jedoch mit den Optionen *ocsp* und *crl* sowie der Unteroption *critical* anpassen.

Mit der folgenden Optionsliste lassen sich Sperrinformationen immer einbetten, wobei eine CRL nur abgerufen wird, sofern OCSP keine gute Antwort liefert:

```
ocsp={critical=false source={url={http://ocsp.acme.com/}}} ↵
 crl={critical=true  source={url={ http://crl.acme.com/}}}
```

Beachten Sie, dass die Optionen *ocsp* und *crl* nur die Einbettung von Sperrinformationen für das Signaturzertifikat, nicht aber für eventuell beteiligte CA- oder TSA-Zertifikate steuern.

7.5 Zeitstempel

7.5.1 Konfiguration von Zeitstempeln

Eine digitale Signatur kann Datums- und Zeitinformationen von einem vertrauenswürdigen Zeitstempel-Server enthalten, auch *Time Stamp Authority* (TSA) genannt. Im Gegensatz zu der Zeit des signierenden Computers (die leicht manipuliert werden kann), stellt ein Zeitstempel von einem vertrauenswürdigen Server eine signierte und zuverlässige Quelle für den Zeitpunkt der Signatur dar. PLOP DS unterstützt Zeitstempel gemäß RFC 3161, RFC 5816 und ETSI EN 319 422. Da die Zeitstempelanforderung einen Hash der erzeugten Signatur enthält, bestätigt der Zeitstempel, dass die Signatur zu einem bestimmten Zeitpunkt erstellt wurde. Der Zeitstempel wird in die erzeugte PDF-Signatur eingebettet.

Abhängig von der ausgewählten TSA sollten Sie bei der Erstellung von Zeitstempeln folgende Konfigurationsaspekte beachten:

- Die wichtigste Information ist die Netzwerkadresse der TSA. Diese kann mit der Unteroption *url* der Unteroption *source* übergeben werden. Alternativ kann sie der digitalen ID des Unterzeichners entnommen werden (siehe »Zeitstempel-Erweiterung in digitalen IDs«, Seite 124).
- Um der TSA zu vertrauen, muss die CA, die das TSA-Zertifikat ausgestellt hat, als vertrauenswürdig anerkannt werden. Das CA-Zertifikat der TSA muss bei der Prüfung der Signatur genauso wie andere CA-Zertifikate behandelt werden; für weitere Informationen siehe »Konfigurieren von vertrauenswürdigen Stammzertifikaten für alle Zertifikatsketten«, Seite 130. Dies ist besonders wichtig bei LTV-fähigen Signaturen. Wenn Sie mit einer TSA unter einer der AATL-Hierarchien arbeiten (siehe »Adobe Approved Trust List (AATL)«, Seite 93), ist Acrobat der Aussteller oder die Kette der Aussteller des TSA-Zertifikats als vertrauenswürdiger Stamm *(Trusted Root)* bekannt. Es kann jedoch nötig sein, das CA-Zertifikat der TSA in der Option *certfile* an PLOP DS zu übergeben.
- Die TSA kann vom Client eventuell verlangen, einen bestimmten Hash-Algorithmus zum Erzeugen der Zeitstempelanforderung zu verwenden. Standardmäßig verwendet PLOP DS den Algorithmus SHA-256, der für alle modernen TSAs funktioniert. Eine andere Hash-Funktion kann mit der Unteroption *hash* übergeben werden. Beachten Sie, dass der in der Zeitstempelsignatur verwendete Hash-Algorithmus nicht angegeben werden kann, da sie vollständig von der TSA kontrolliert wird.
- Manche TSAs sind frei zugänglich, einige kommerzielle TSAs benötigen zur Zugriffsberechtigung allerdings Benutzernamen und Kennwort. Unberechtigter Zugriff führt zu etwa folgender Fehlermeldung:

  ```
  Network response from URL 'https://timestamp.acme.com/tsa' has bad status code 401
  ('Unauthorized')
  ```

  Parameter für die Authentisierung können als Teil der URL oder mit den Unteroptionen *username/password* der Netzwerk-Unteroption *source* übergeben werden.
- Falls die TSA SSL-Zugriff (d.h. *https)* erfordert, muss das SSL-Stammzertifikat des Servers mit den Optionen *sslcertdir/sslcertfile* übergeben werden. Andernfalls wird eine Fehlermeldung ähnlich der folgenden ausgegeben:

  ```
  Document time-stamp request to 'https://timestamp.acme.com/tsa' failed
  ('Peer certificate cannot be authenticated with given CA certificates')
  ```

Sie können die Prüfung des erforderlichen Server-Zertifikats auch mit der Option *sslverifypeer=false* überspringen, vorausgesetzt, Sie sind sich dabei der Auswirkungen auf die Sicherheit bewusst.

- Manche TSAs benötigen eine explizite Richtlinien-OID (Object Identifier), die mit der Option *policy* übergeben werden kann. Der entsprechende Wert der OID muss mit der TSA abgeklärt werden. Der Richtlinien-OID wird in Acrobat im Fenster *Unterschrifteneigenschaften* unter *Erweiterte Eigenschaften...* angezeigt.

7.5.2 Signaturen mit eingebettetem Zeitstempel

Hinweis *Signaturen mit eingebettetem Zeitstempel werden für* `engine=mscapi` *nicht unterstützt.*

Genehmigungs- und Zertifizierungssignaturen können optional eingebettete Zeitstempel enthalten. Signaturen mit eingebettetem Zeitstempel werden ab Acrobat 7 unterstützt.

Zeitstempel-Erweiterung in digitalen IDs. Eine digitale ID kann die Erweiterung *TimeStamp* mit der URL einer TSA enthalten. Sie erleichtert das Signieren mit eingebetteten Zeitstempeln, da auf die Übergabe von TSA-Details verzichtet werden kann. Die Erweiterung *TimeStamp* ist normalerweise in von AATL *(Adobe Approved Trust List)* zur Verfügung gestellten Zertifikaten enthalten (siehe »Adobe Approved Trust List (AATL)«, Seite 93). Einige AATL-Anbieter bieten eine begrenzte Anzahl von Zeitstempeln kostenlos an.

Wenn die *TimeStamp*-Erweiterung vorhanden ist und eine URL enthält, die keine Authentisierung erfordert, versucht PLOP DS zur Erzeugung eines Zeitstempels auf die angegebene TSA zuzugreifen. In diesem Fall ist die Übergabe der Unteroption *url* zur Erzeugung eines Zeitstempels nicht erforderlich. Für eine TSA mit Authentisierung müssen Sie jedoch die vollständigen TSA-Informationen explizit in einer Optionsliste übergeben (siehe die Beispiele unten), selbst wenn die TSA in der *TimeStamp*-Erweiterung angegeben ist.

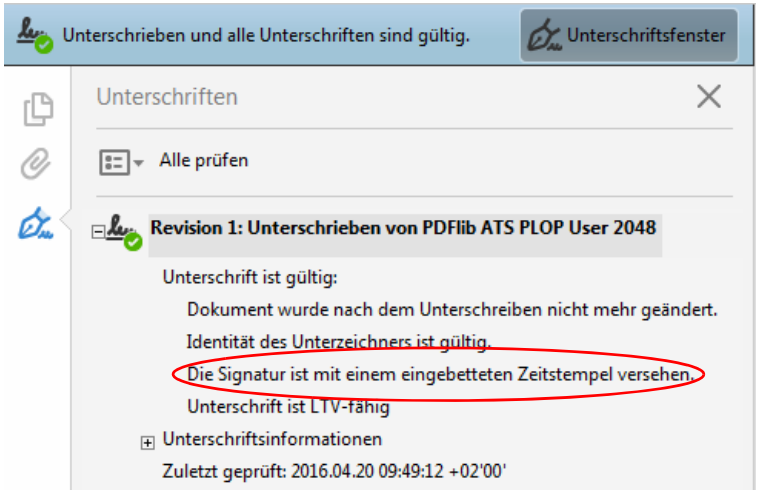


Abb. 7.9
Signaturen mit eingebettetem Zeitstempel in Acrobat

Beispiele für Zeitstempel-Optionslisten. In den folgenden Beispielen wird nur der Teil der Optionsliste aufgeführt, der für die Einbettung des Zeitstempels relevant ist. Je nach Bedarf müssen Sie weitere Signaturoptionen hinzufügen.

Signatur mit einem Zeitstempel versehen, der von der TSA über die angegebene URL angefordert wurde, unter Verwendung des Standard-Hash-Algorithmus SHA-256:

```
timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

Signatur mit einem Zeitstempel versehen, wobei die TSA zur Ermittlung des Zeitstempels Benutzernamen und Kennwort benötigt:

```
timestamp={source={url={http://timestamp.acme.com/tsa} username=demo password=demo}}
```

Signatur mit einem Zeitstempel versehen, wobei die TSA Digest-Authentisierung erfordert:

```
timestamp={source={url={http://timestamp.acme.com/tsa} httpauthentication=digest ↵
        username=demo password=demo }}
```

Falls über SSL auf die TSA zugegriffen werden muss, so müssen Sie das SSL-Stammzertifikat des Servers mit den Optionen *sslcertdir/sslcertfile* übergeben. Ist das SSL-Zertifikat des Servers nicht verfügbar, können Sie die Server-Authentisierung mit der Option *sslverifypeer=false* überspringen, vorausgesetzt, Sie sind sich dabei der Auswirkungen auf die Sicherheit bewusst:

```
timestamp={source={url={https://timestamp.acme.com/tsa}} sslverifypeer=false}
```

Einbettung eines Zeitstempels in die Signatur mit der in der digitalen ID des Unterzeichners vorhandenen URL und Abbruch mit einer Fehlermeldung, wenn in der digitalen ID keine *TimeStamp*-Erweiterung vorhanden ist:

```
timestamp={source={}}
```

oder äquivalent

```
timestamp={}
```

Keine Einbettung eines Zeitstempels, selbst wenn die *TimeStamp*-Erweiterung in der digitalen ID vorhanden ist:

```
timestamp=none
```

7.5.3 Zeitstempelsignaturen auf Dokumentebene

Zeitstempel auf Dokumentebene wurden mit PAdES Teil 4 eingeführt und in ISO 32000-2 aufgenommen. Zeitstempel auf Dokumentebene werden ab Acrobat X unterstützt.

Signaturen mit eingebettetem Zeitstempel versus Zeitstempelsignaturen auf Dokumentebene. Ähnlich wie eine Signatur mit eingebettetem Zeitstempel liefert auch eine Zeitstempelsignatur auf Dokumentebene Statusinformationen, die sich auf einen bestimmten Zeitpunkt beziehen. Im ersten Fall ist der Zeitstempel jedoch ein Attribut der Hauptsignatur, während ein Zeitstempel auf Dokumentebene selbst eine gültige Signatur darstellt. Er erfordert keine digitale ID, da keine unterzeichnende Person oder

Körperschaft beteiligt ist. Stattdessen werden die Zeitstempel auf Dokumentebene über ein Netzanfrage an eine *Time Stamp Authority* (TSA) erstellt. Zeitstempel auf Dokumentebene gewährleisten, dass ein bestimmtes Dokument zu dem im Zeitstempel angegebenen Zeitpunkt existiert hat.

Hinweis *Zeitstempelsignaturen auf Dokumentebene werden für* `engine=mscapi` *nicht unterstützt.*

Beispiele für Optionslisten mit Zeitstempeln auf Dokumentebene. In den folgenden Beispielen wird die vollständige Liste von Signaturoptionen aufgeführt, die für die Erstellung eines Zeitstempels benötigt werden. Da kein Signaturzertifikat benötigt wird, sind keine weiteren Optionen erforderlich.

Hinzufügen eines Zeitstempels auf Dokumentebene, der von der TSA über die angegebene URL angefordert wird, unter Verwendung des Standard-Hash-Algorithmus SHA-256:

```
doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

Hinzufügen eines Zeitstempels auf Dokumentebene von einer TSA, die Benutzernamen und Kennwort benötigt:

```
doctimestamp={source={url={http://timestamp.acme.com/tsa}} username=demo password=demo}
```

Hinzufügen eines Zeitstempels auf Dokumentebene von einer TSA, die Digest-Authentisierung erfordert:

```
doctimestamp={source={url={http://timestamp.acme.com/tsa} httpauthentication=digest ↵
        username=demo password=demo}}
```

pCOS. Zeitstempelsignaturen auf Dokumentebene werden in pCOS als *signaturefields[...]/sigtype=doctimestamp* ausgegeben.

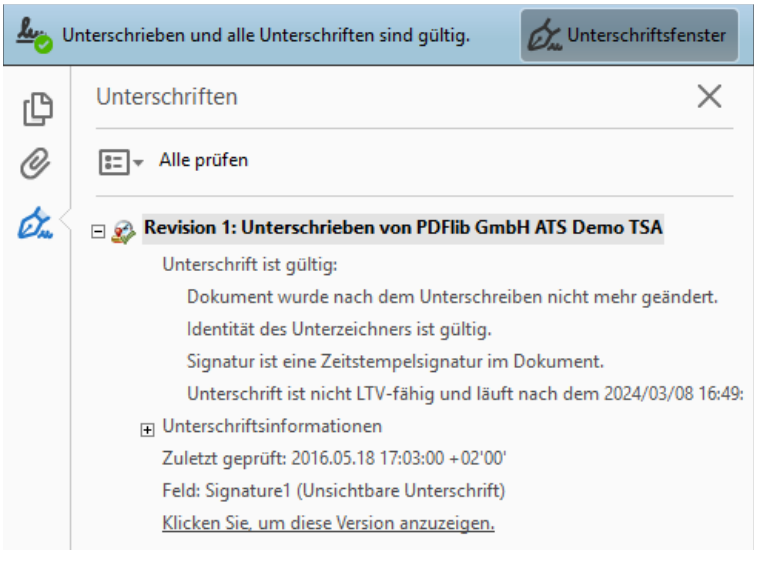


Abb. 7.10 Zeitstempel auf Dokumentebene in Acrobat

7.5.4 Fehlerbehebung und nicht unterstütze TSAs

Übergroße Zeitstempel-Anforderungen. PLOP DS muss die Größe eines Zeitstempels im Voraus bekannt sein. Es verwendet einen internen Wert für die maximale Größe der von der TSA erhaltenen Zeitstempel-Anforderung. Übersteigt die Zeitstempel-Anforderung dieses Maximum, tritt folgender Fehler auf:

```
Not enough space reserved for signature contents (reserved XXX bytes, need YYY bytes)
```

In diesem Fall können Sie den Maximalwert mit der Signaturoption *timestampsize* erhöhen. Der interne Standardwert von *timestampsize* ist in Tabelle 8.8, Seite 147 aufgeführt.

In den unten aufgeführten Fällen können mit einer TSA keine PDF-Dokumente mit PLOP DS signiert werden.

Attributzertifikate. Attributzertifikate werden von PLOP DS nicht unterstützt. Werden sie von einer TSA verwendet, gibt PLOP DS folgende Fehlermeldung aus:

```
Time-stamp authority 'http://adobe-tsa.entrust.net/TSS/HttpTspServer'
uses unsupported protocol ('wrong tag')
```

Ein konkreter Einsatzfall von Attributzertifikaten ist für das *Time Auditing Certificate* (TAC) einer TSA. Einige TSA-Produkte verwenden die neue CMS-Syntax gemäß RFC 2630 zur Kodierung der TAC, was von PLOP DS nicht unterstützt wird. Sie können jedoch so konfiguriert werden, dass das TAC mit alternativen Methoden, wie dem Hinzufügen des TAC in ein signiertes Attribut gemäß RFC 3126 kodiert wird.

Erweiterung »key usage« nicht als kritisch markiert. Für das Zeitstempel-Protokoll muss das TSA-Zertifikat die Erweiterung *Extended Key Usage* mit dem Wert *timestamping* enthalten, wobei die Erweiterung als kritisch markiert sein muss. Ist die Erweiterung im TSA-Zertifikat zwar vorhanden, aber nicht als kritisch markiert, wird die Signatur von Acrobat als ungültig zurückgewiesen.

PLOP DS weist mit einer solchen TSA erzeugte Zeitstempel mit der folgenden Fehlermeldung zurück:

```
Signature verification of time-stamp failed: certificate verify error:
Verify error:unsupported certificate purpose
```

Der Versuch, in Acrobat einen Zeitstempel auf Dokumentebene mit Hilfe eines TSA-Zertifikats zu erzeugen, in dem das Feld *Extended Key Usage* nicht als kritisch markiert ist, führt zu folgender Fehlermeldung:

```
Error encountered while signing:
Certificate is not valid for the usage
```

Es ist in Acrobat zwar möglich, eine Zertifizierungs- oder Genehmigungssignatur mit einem eingebetteten Zeitstempel mit Hilfe einer solchen TSA zu erzeugen, aber bei der Validierung wird der Zeitstempel in der resultierenden Signatur mit folgender Fehlermeldung zurückgewiesen:

```
The signature includes an embedded timestamp but it is invalid
```

Authenticode-Zeitstempel-Server. Authenticode ist ein Zeitstempel-Protokoll von Microsoft, das hauptsächlich zur Signierung von Code eingesetzt wird. Da Authenticode auf dem älteren RFC 2985/PKCS#9 statt auf RFC 3161 basiert, wird es in PDF und PLOP DS nicht unterstützt.

PLOP DS weist mit einer Authenticode-TSA erzeugte Zeitstempel mit einer Fehlermeldung ähnlich der folgenden zurück:

```
Unexpected content type 'text/html;charset=ISO-8859-1' in reply to time-stamp request to
URL 'http://timestamp.entrust.net/TSS/AuthenticodeTS'
(expected content type 'application/timestamp-reply')
```

oder

```
Unexpected content type 'application/timestamp-query' in reply to time-stamp request to
URL 'http://timestamp.verisign.com/scripts/timstamp.dll'
(expected content type 'application/timestamp-reply')
```

Der Versuch, eine Authenticode-TSA mit Acrobat zu verwenden, führt zu folgender Fehlermeldung:

```
Error encountered while signing:
Error encountered while BER decoding
```

7.6 Langzeitvalidierung (LTV)

7.6.1 Langzeitvalidierung und Acrobat

Bei der Langzeitvalidierung *(Long-Term Validation, LTV)* kann die Signatur auch dann noch validiert werden, wenn das Signaturzertifikat abgelaufen ist oder gesperrt wurde, was bei der Archivierung von signierten Dokumenten über einen langen Zeitraum ein wichtiger Aspekt ist. Das Konzept der Langzeitvalidierung wird in PAdES Teil 4 (ETSI TS 102 778-4) behandelt und in Acrobat XI/DC unterstützt.

Um eine Signatur LTV-fähig zu machen, müssen die vollständige Zertifikatskette sowie Sperrinformationen für alle beteiligten Zertifikate, die sogenannten Prüfinformationen, in die Signatur oder in einen DSS eingebettet werden (siehe Abschnitt 7.3.3, »Document Security Store (DSS)«, Seite 110). Da für LTV weitere Signatur-bezogene Daten eingebettet werden, sind die signierten Dokumente in der Regel größer als nicht LTV-fähige Signaturen.:

Hinweis *LTV-fähige Signaturen werden für* `engine=mscapi` *nicht unterstützt.*

LTV-fähige Signaturen sollten einen eingebetteten Zeitstempel enthalten, wobei dies nicht zwingend vorgeschrieben ist. Sie können die Lebensdauer einer LTV-fähigen Signatur verlängern, indem Sie eine Zeitstempelsignatur auf Dokumentebene hinzufügen, bevor eines der beteiligten Zertifikate abläuft oder gesperrt wird.

Der LTV-Status ist nicht absolut definiert, sondern in Bezug auf eine Menge von vertrauenswürdigen Stammzertifikaten. Abhängig von der Konfiguration kann eine Signatur in einer Konfiguration als LTV-fähig gelten, in einer anderen Konfiguration aber nicht. Wenn Sie beispielsweise unterschiedliche vertrauenswürdige Stammzertifikate in PLOP DS und in Acrobat konfigurieren, kann sich ihr LTV-Status ändern.

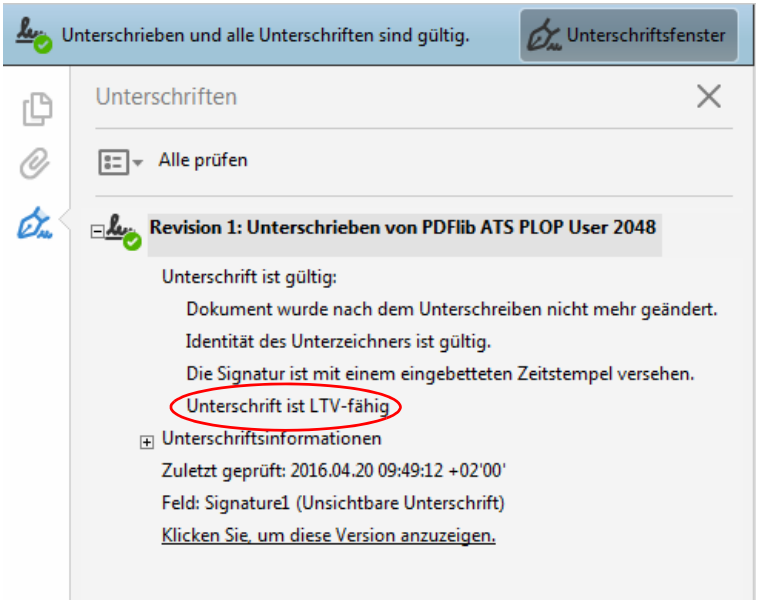


Abb. 7.11
Signatur für die Langzeitvalidierung in Acrobat

LTV-Status in Acrobat. In Acrobat XI/DC wird die Statuszeile »Unterschrift ist LTV-fähig« oder »Unterschrift ist nicht LTV-fähig und läuft nach dem...ab« im Unterschriftsfenster angezeigt (siehe Abbildung 7.11). Beachten Sie bei der Statuszeile zu LTV folgendes

- Das oder die Stammzertifikate für alle beteiligten Zertifikate müssen in Acrobat als vertrauenswürdig konfiguriert sein (siehe »Vertrauenswürdige Stammzertifikate in Acrobat (Trusted Root Certificates)«, Seite 93).
- Jede gültige Signatur kann in Acrobat als LTV-fähig angezeigt werden, wenn die unmittelbaren CA-Signaturzertifikate zu den vertrauenswürdigen Stammzertifikaten hinzugefügt werden. Wenn die Konfiguration nicht berücksichtigt wird, kann dies bezüglich des LTV-Status zu Verwirrung führen. Es bedeutet auch, dass mit einem selbst signierten Zertifikat erstellte Signaturen als LTV-fähig behandelt werden, wenn das Zertifikat zu den vertrauenswürdigen Stammzertifikaten hinzugefügt wird.
- In Acrobat XI/DC ist ein eingebetteter Zeitstempel für die Langzeitvalidierung nicht zwingend erforderlich. Ist ein Zeitstempel eingebettet, benötigt Acrobat keine Prüfinformationen für das TSA-Zertifikat. PLOP DS ist hier strikter und verlangt die vollständigen Prüfinformationen für das TSA-Zertifikat. Acrobat verwendet keine eingebettete OCSP-Antwort für das TSA-Zertifikat, es sei denn, es wurde nur wenige Minuten vor der Validierungszeit erstellt.
- Acrobat XI/DC unterstützt nur die Hash-Funktion SHA-1 für OCSP-Antworten (siehe »OCSP-Konfiguration«, Seite 118). Wenn eine andere Hash-Funktion verwendet wird, zeigt Acrobat den LTV-Status unter Umständen nicht korrekt an, obwohl die Prüfinformationen vollständig verfügbar sind.
- Folgende Einstellung sollten Sie in Acrobat nicht aktivieren, da sonst der LTV-Status nicht angezeigt wird: *Voreinstellungen, Unterschriften, Überprüfung, Weitere..., Zeitpunkt der Überprüfung, Unterschriften prüfen anhand folgendem Kriterium: Aktuelle Uhrzeit.*
- Beim Wiederherstellen einer früheren Revision kann der LTV-Status verloren gehen; für weitere Informationen siehe »Wiederherstellen früherer Revisionen eines signierten Dokuments«, Seite 112.

7.6.2 LTV-fähige Signaturen in PLOP DS

Mit der folgenden Option erzeugt PLOP DS eine LTV-fähige Signatur, falls alle Prüfinformationen beschafft werden können. Andernfalls schlägt die Funktion fehl und es wird keine Signatur erzeugt:

```
ltv=full
```

Diese Option allein kann noch keine LTV-fähigen Signaturen gewährleisten, sondern überprüft nur, ob alle Voraussetzungen dafür erfüllt sind. Fehlt ein Zertifikat oder Prüfinformationen, gibt PLOP DS eine Fehlermeldung aus. Sie sollten deshalb alle Fehlermeldungen genau analysieren.

Die Standardeinstellung *ltv=try* bedeutet, dass alle verfügbaren Sperrinformationen in das signierte Dokument eingebettet werden, aber der Aufruf der Signatur nicht fehlschlägt, wenn die Prüfinformationen für den LTV-Status nicht ausreichen.

Konfigurieren von vertrauenswürdigen Stammzertifikaten für alle Zertifikatsketten. Um alle beteiligten Zertifikate vollständig zu validieren, benötigt PLOP DS für alle Zertifikate Vertrauensanker. Die genaue Anzahl ist abhängig von der PKI-Konfigur-

ation. Vertrauenswürdige Stammzertifikate müssen mit der Option *rootcertdir* oder *rootcertfile* übergeben werden. Dies umfasst zumindest das Zertifikat der Stammzertifizierungsstelle *(Root CA)* am Anfang der Kette für das Signaturzertifikat. Weitere Stammzertifikate, z.B. für die TSA, werden eventuell benötigt, sofern sich nicht eine einzige Stammzertifizierungsstelle am Anfang aller beteiligten Zertifikatsketten befindet.

Konfigurieren von CA-Zwischenzertifikaten. Die verbleibende Zertifikatskette, (d.h. alle zwischengeschalteten CAs zwischen dem Stamm- und Signaturzertifikat oder anderen beteiligten Zertifikaten) muss vorhanden sein, damit PLOP DS sie in die Signatur einbetten kann. CA-Zertifikate für das Signaturzertifikat sowie andere beteiligte Zertifikate wie ein OCSP-Responder- oder TSA-Zertifikat werden an folgenden Orten gesucht:

- CA-Zertifikate können mit der Option *certfile* übergeben werden.
- (Nicht bei *engine=mscapi)* CA-Zertifikate für das Signaturzertifikat können in der PKCS#12-Datei enthalten sein, die die digitale ID des Unterzeichners enthält.
- (Nur bei *engine=mscapi)* CA-Zertifikate werden im Zertifikatspeicher von Windows gesucht.
- Ein Zertifikat kann eine Erweiterung namens *Authority Info Access (AIA)* mit der Zugriffsmethode *caIssuers (Certification Authority Issuer)* gemäß RFC 3280 enthalten. Diese Erweiterung enthält eine oder mehrere URLs, über die das Zertifikat der CA heruntergeladen werden kann, welche das Signaturzertifikat und eventuelle CA-Zwischenzertifikate ausgestellt hat. Die Protokolle *http*, *https* und *ftp* werden unterstützt.

Ein Zertifikat kann das LDAP-Protokoll in der AIA-Erweiterung angeben, was von PLOP DS derzeit nicht unterstützt wird. In diesem Fall können Sie mit einem LDAP-Browser[1] das CA-Zertifikat manuell über LDAP herunterladen und es an die oben genannten Optionen übergeben. Dies muss nur einmalig für ein Signaturzertifikat durchgeführt werden.

Welche CA-Zertifikate muss ich konfigurieren? Die genauen Anforderungen für die LTV-Fähigkeit hängen von der PKI-Konfiguration ab. In vielen Fällen reichen die folgende Schritte aus:

- Zertifikate, die von vielen kommerziellen Zertifizierungsstellen ausgestellt werden, enthalten die AIA-Erweiterung mit der *caIssuers*-Zugriffsmethode. Damit kann PLOP DS automatisch die CA-Zertifikatskette für das Signaturzertifikat herunterladen. Nur das vertrauenswürdige CA-Stammzertifikat muss mit der Option *rootcertdir* oder *rootcertfile* übergeben werden.
  Ist die AIA-Erweiterung mit der *caIssuers*-Zugriffsmethode im Signaturzertifikat nicht vorhanden, können Sie das/die benötigte(n) Stammzertifikat(e) im Allgemeinen von der Website der Zertifizierungsstelle (CA) herunterladen.
- Zertifikate von TSAs und OCSP-Respondern werden automatisch heruntergeladen. Wurden diese Zertifikate oder weitere Zwischenzertifikate von einer anderen Stammzertifizierungsstelle als der des Signaturzertifikats ausgestellt, müssen Sie das Stammzertifikat mit der Option *rootcertdir* oder *rootcertfile* übergeben.
- CRLs werden häufig von der selben CA signiert, die auch das überprüfte Zertifikat ausgestellt hat. Wurde die CRL jedoch von einer anderen CA signiert, müssen Sie das zugehörige CA-Zertifikat mit der Option *certfile* übergeben, da es nicht automatisch heruntergeladen werden kann. Wurde das Zertifikat zum Signieren einer CRL von ei-

1. Zum Beispiel der kostenlose Softerra LDAP-Browser, der unter folgender Adresse verfügbar ist: www.ldapbrowser.com/

ner anderen Stammzertifizierungsstelle ausgestellt als das Signaturzertifikat (z.B. die CRL für eine TSA, die Teil einer anderen PKI ist), müssen Sie das Stammzertifikat mit den Optionen *rootcertdir* oder *rootcertfile* übergeben.

Mit *validate=full* oder *ltv=full* gibt PLOP DS die Fehlermeldung »*unable to get local issuer certificate*« aus, falls ein erforderliches CA-Zertifikat fehlt. In diesem Fall müssen Sie das fehlende Zertifikat mit einer der Optionen *rootcertdir, rootcertfile* oder *certfile* übergeben. Die Fehlermeldung:

```
Certificate verification failure for certificate with subject '...':
self signed certificate in certificate chain
```

tritt auf, wenn ein vertrauenswürdiges selbst signiertes Zertifikat in der Option *certfile* statt in der Option *rootcertfile* oder *rootcertdir* übergeben wurde.

Sperrinformationen für das Signaturzertifikat. Sperrinformationen für das Signaturzertifikat müssen auf eine der folgenden Arten übergeben werden:

- OCSP mittels der *AIA*-Erweiterung im Zertifikat des Unterzeichners oder der Option *ocsp*.
- CRL mittels der *CRLdp*-Erweiterung im Zertifikat des Unterzeichners oder der Option *crl*. Bestehende CRLs können mit den Optionen *crldir* und *crlfile* übergeben werden.

Mit der Unteroption *critical* der Optionen *ocsp* und *crl* kann sichergestellt werden, dass eine Signatur nur dann erstellt wird, wenn für das Signaturzertifikat die OCSP- oder CRL-Informationen erfolgreich beschafft werden konnten. Für weitere Informationen siehe Abschnitt 7.4, »Sperrinformationen zu Zertifikaten«, Seite 117.

Sperrinformationen für andere beteiligte Zertifikate. Sperrinformationen für die Zertifikate aller CAs in der Zertifikatskette sowie für die Zertifikate aller für das Signieren von CRLs und OCSP-Antworten verwendeten CAs müssen ebenfalls verfügbar sein. Dabei gelten folgende Ausnahmen, bei denen keine Sperrinformationen benötigt werden:

- an die Optionen *rootcertdir* oder *rootcertfile* übergebene Zertifikate der Stammzertifizierungsstelle;
- das Zertifikat eines OCSP-Responders, sofern es die Erweiterung *id-pkix-ocsp-nocheck* enthält (was normalerweise der Fall ist).

Sperrinformationen für andere Zertifikate als die Signaturzertifikate müssen auf eine der folgenden Arten übergeben werden:

- OCSP mittels der *AIA*-Erweiterung im Zertifikat.
- CRL mittels der *CRLdp*-Erweiterung im Zertifikat. Bestehende CRLs können mit den Optionen *crldir* und *crlfile* übergeben werden.

Beispiele für LTV-Optionslisten. Für das erste Beispiel gehen wir davon aus, dass die PKI folgendermaßen aufgebaut ist:

- die digitale ID des Unterzeichners enthält die CA-Zertifikatskette in der PKCS#12-Datei oder alle Zertifikate außer dem Stammzertifikat enthalten die AIA-Erweiterung mit der *caIssuers*-Zugriffsmethode;
- die digitale ID des Unterzeichners sowie alle CA-Zertifikate in der Kette außer dem Stammzertifikat enthalten die AIA-Erweiterung mit der *ocsp*-Zugriffsmethode oder die *CRLdp*-Erweiterung;
- das Zertifikat des OCSP-Responders enthält die Erweiterung *id-pkix-ocsp-nocheck*.

In diesem Fall ist nur die Option *rootcertfile* erforderlich (zusätzlich zu Optionen für die digitale ID), um die LTV-Fähigkeit zu erreichen. Mit der Option *ltv=full* kann sichergestellt werden, dass Verletzungen der LTV-Anforderungen erkannt werden und keine Signatur erstellt wird, falls die LTV-Fähigkeit nicht erreicht werden kann:

```
digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=root1.pem
```

Um einen Zeitstempel einzubetten, müssen für die LTV-Fähigkeit auch die Sperrinformationen für die TSA verfügbar sein. Im Idealfall enthält das TSA-Zertifikat auch die *AIA*-Erweiterung mit der *ocsp*-Zugriffsmethode oder die *CRLdp*-Erweiterung und geht auf die selbe Stammzertifizierungsstelle zurück wie das Signaturzertifikat. In diesem Fall werden keine weiteren Optionen für die LTV-Fähigkeit benötigt:

```
digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=root1.pem timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

Ist die TSA jedoch in einer anderen Stammzertifizierungsstelle verankert, müssen Sie mit der Option *rootcertfile* auch den TSA-Stamm übergeben (unter der Annahme, dass die Datei *root1+2.pem* die zwei erforderlichen Stammzertifikate in PEM-Kodierung enthält):

```
digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=root1+2.pem timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

7.7 Die Signaturstandards CAdES und PAdES

7.7.1 CMS- und CAdES-Signaturen

Das *European Telecommunications Standards Institute* (ETSI)[1] gibt eine Reihe von Standards zur digitalen Signatur heraus, um digitale Signaturen unter den EU-Mitgliedsstaaten zu vereinheitlichen. ETSI-Standards sind auch in anderen Teilen der Welt recht einflussreich. Sie werden im Standard ISO 32000-2 für PDF-2.0 referenziert und wurden in verschiedene RFCs aufgenommen.

CMS- und CAdES-Signaturen. PDF-Signaturen basierten lange Zeit auf CMS *(Cryptographic Message Syntax)*. Dieses Format ist in RFC 5652 spezifiziert und in Internet-Protokollen weit verbreitet. In PDF verwenden CMS-Signaturen den Subfilter-Eintrag *adbe.pkcs7.detached* oder einige andere veraltete Einträge im Signatur-Dictionary. CMS-Signaturen können mit allen Acrobat-Versionen validiert werden.

CAdES *(CMS Advanced Electronic Signatures)* ist in ETSI TS 101 733 spezifiziert (technisch äquivalent zu RFC 5126) und fügt einige zusätzliche CMS-Funktionen hinzu. Es schützt vor allem vor einem als Zertifikat-Substitution bekannten Angriffsszenario, und zwar durch einen Verweis auf das Signaturzertifikat in der Signatur (unter Verwendung des Attributs *signing-certificate-v2)*. In PDF benötigen CAdES-Signaturen den Subfilter-Eintrag *ETSI.CAdES.detached* im Signatur-Dictionary. Erstellung und Validierung von CAdES-Signaturen werden ab Acrobat X unterstützt.

pCOS. CAdES-Signaturen werden in pCOS als *signaturefields[...]/cades=true* ausgegeben.

PAdES-Signaturen. PAdES *(PDF Advanced Electronic Signatures)* ist in ETSI TS 102 778 spezifiziert. Es wendet CAdES auf PDF an, indem weitere Optionen und Einschränkungen für PDF-Signaturen gemäß PDF 1.7 (ISO 32000-1) hinzugefügt werden. PAdES legt auch zusätzliche PDF-Datenstrukturen fest, die in PDF 2.0 (ISO 32000-2) enthalten sind. PAdES besteht aus mehreren Teilen (hier nicht relevante Teile bleiben unberücksichtigt).

- PAdES Teil 2 ist in ETSI TS 102 778-3 PAdES Basic spezifiziert. Es basiert auf CMS und ist konform zu ISO 32000-1, wobei einige seiner Optionen nicht erlaubt sind, um die Signaturen zu stärken. In PAdES Basic muss z.B. *ByteRange* das gesamte Dokument abdecken, um eine Sicherheitslücke in der ursprünglichen ISO 32000-1-Definition zu schließen.
- PAdES Teil 3 ist in ETSI TS 102 778-3 PAdES Enhanced spezifiziert. Es basiert auf CAdES und definiert die zwei Profile BES *(Basic Electronic Signature)* und EPES *(Explicit Policy-based Electronic Signature)*. EPES erweitert BES um einen Richtlinien-Identifikator und einen optionalen *Commitment Type Indicator* für die Signatur. Das Attribut *policy* legt die Signaturrichtlinie fest, die für die Erstellung der Signatur gilt. Das Attribut *commitment-type* kann alternativ zum Eintrag *Reason* im Signatur-Dictionary verwendet werden. CAdES definiert eine Reihe von generischen Commitment-Typen wie *proof of origin*, *proof of receipt* oder *proof of approval*.
- PAdES Teil 4 ist in ETSI TS 102 778-4 PAdES Long-Term spezifiziert und definiert die Methoden zur Langzeitvalidierung; für weitere Informationen siehe Abschnitt 7.6, »Langzeitvalidierung (LTV)«, Seite 129. Teil 4 führt Zeitstempel auf Dokumentebene

1. ETSI-Standards sind unter folgender Adresse kostenlos verfügbar: www.etsi.org/standards

sowie den DSS ein (siehe Abschnitt 7.3.3, »Document Security Store (DSS)«, Seite 110). Die in PAdES Teil 4 definierten Konzepte können auf Signaturen vom Typ PAdES Teil 2 und Teil 3 angewendet werden, d.h. PAdES-LTV kann auf CMS oder CAdES basieren.

PAdES-Signaturstufen. Verschiedene Stufen von PAdES-Signaturen wurden definiert, um den Lebenszyklus einer Unterschrift abdecken zu können. Die folgenden grundlegenden PAdES-Signaturstufen sind in ETSI EN 319 142-1 spezifiziert:

- Einfache Signatur: PAdES-Signaturstufe B-B (früher PAdES-B) ist der Basisbaustein von PAdES-Signaturen. Diese Stufe unterstützt Signaturen mit und ohne Richtlinien-Identifikator, d.h. EPES und BES.
- Signatur mit Zeitstempel: PAdES-Signaturstufe B-T fügt einen Signatur-Zeitstempel zur Signaturstufe B-B hinzu, um zu beweisen, dass die Signatur zu einem bestimmten Datum und Zeitpunkt existierte.
- Signatur mit Material zur Langzeitvalidierung: PAdES-Signaturstufe B-LT fügt Prüfinformationen zur Stufe B-T hinzu, um die langfristige Verfügbarkeit des Prüfmaterials zu gewährleisten.
- Signaturen mit Langzeitvalidierung und Integrität von Validierungsmaterial: PAdES-Signaturstufe B-LTA fügt einen Zeitstempel *(Archive)* auf Dokumentebene zur Signaturstufe B-LT hinzu, um die langfristige Verfügbarkeit und Integrität von Prüfinformationen zu gewährleisten. Zeitstempel können wiederholt hinzugefügt werden, z.B. wenn kryptografische Algorithmen und Schlüssellängen nicht mehr als stark genug betrachtet werden.

ETSI EN 319 142-2 »Part 2: Extended PAdES signatures« definiert erweiterte Signaturprofile:

- PAdES-Signaturstufe E-BES definiert die grundlegenden Anforderungen für digitale Signaturen in PDF.
- PAdES-Signaturstufe E-EPES baut auf PAdES E-BES auf. Sie fügt einen Richtlinien-Identifikator und einen optionalen *Commitment Type Indicator* hinzu.
- PAdES-Signaturstufe E-LTV kann entweder auf E-BES oder E-EPES aufbauen. Sie fügt einen Document Security Store (DSS) und Zeitstempelsignaturen auf Dokumentebene hinzu. Damit kann eine vorhandene Signatur erweitert werden, um die langfristige Gültigkeit der Signatur zu erhalten.

CAdES- und PAdES-Unterstützung in Acrobat. Standardmäßig sind Acrobat-Signaturen konform zu PAdES Teil 2 (PAdES Basic). Acrobat X/XI/DC können Signaturen gemäß PAdES Teil 3 BES erstellen, wenn sie folgendermaßen für CAdES konfiguriert werden:

- Acrobat XI/DC: *Bearbeiten, Voreinstellungen, Unterschriften, Erstellung und Erscheinungsbild, Weitere..., Standard-Signierungsformat: CAdES-Äquivalent*
- Acrobat X: *Bearbeiten, Voreinstellungen, Sicherheit, Erweiterte Voreinstellungen..., Erstellung, Standardsignatur - Signaturformat: CAdES-Äquivalent*.

Da es keine Unterstützung für Richtlinien-Identifikatoren gibt, kann PAdES E-EPES nicht mit Acrobat erzeugt werden. Aber sowohl E-BES als auch E-EPES können mit Acrobat validiert werden.

PAdES Teil 4 wird in Acrobat X/XI/DC mit folgenden Funktionen unterstützt:

- Acrobat XI/DC: Für weitere Statusinformationen zur Langzeitvalidierung siehe »LTV-Status in Acrobat«, Seite 130;
- In Acrobat X/XI/DC: Sie können eine Signatur LTV-fähig machen, indem Sie das Navigationsfenster *Unterschriften* öffnen und im Optionsmenü *Prüfinformationen hinzufügen* auswählen.
- Acrobat X/XI/DC: Zeitstempelsignaturen auf Dokumentebene.

Ab der Veröffentlichung von Acrobat DC mit Oktober 2016 (genauer: Acrobat DC Classic 2015.006.30243 und Acrobat DC Continuous 2015.020.20039) können Sie die PAdES-Signaturstufe anzeigen, indem Sie auf eine Signatur rechtsklicken und *Unterschrifteneigenschaften einblenden...*, *Erweiterte Eigenschaften...* auswählen (siehe Abbildung 7.12).

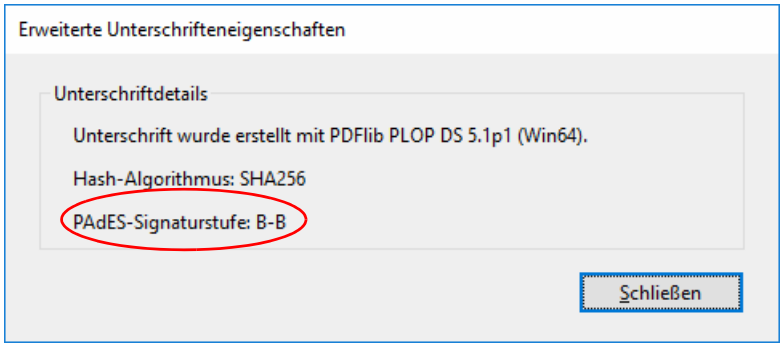


Abb. 7.12
PAdES-Status in Acrobat DC

7.7.2 PAdES-Signaturen mit PLOP DS

PLOP DS unterstützt bei Autoren- und Genehmigungssignaturen alle oben aufgeführten PAdES-Signaturstufen. Der Signaturtyp CMS oder CAdES kann mit der Option *sigtype* ausgewählt werden; Funktionen für die PAdES-Signaturstufen können durch zusätzliche Optionen aktiviert werden. PLOP DS unterstützt standardmäßig CAdES-Signaturen, die konform sind zu PAdES-Signaturstufe B-B. Tabelle 7.7 führt die Optionen auf, die zur Erreichung der PAdES-Signaturstufen mit PLOP DS erforderlich sind.

Hinweis PAdES Teil 3 und Teil 4 werden für `engine=mscapi` *nicht unterstützt.*

Tabelle 7.7 PAdES-Signaturstufen gemäß ETSI EN 319 142-1

PAdES-Signaturstufe	***Optionen von PLOP DS***
PAdES-Signaturstufe B-B, einschließlich E-BES (Basic Electronic Signature) und E-EPES (Explicit Policy Electronic Signature)	*PAdES E-BES:* `sigtype=cades` *(Standardwert)* *PAdES E-EPES: wie PAdES E-BES plus* `policy`
PAdES-Signaturstufe B-T (vertrauenswürdige Zeit für das Bestehen der Signatur)	*wie PAdES-Signaturstufe B-B plus* `timestamp` *mit* `critical=true`
PAdES -Signaturstufe B-LT (Long-Term)	*wie PAdES-Signaturstufe B-T plus* `ltv=full rootcertfile/rootcertdir`[1]
PAdES-Signaturstufe B-LTA (Long-Term mit Archiv-Zeitstempeln)	*wie PAdES-Signaturstufe B-LT plus* `doctimestamp`
PAdES-Signaturstufe E-LTV (Long Term Validation)	*wie PAdES-Signaturstufe B-LTA plus* `dss=true`*, auf ein Dokument mit einer vorhandenen Signatur angewendet*

1. Für die LTV-Fähigkeit können weitere Optionen erforderlich sein, wie zum Beispiel `certfile`, `ocsp`, `crl`*; siehe Abschnitt 7.6.2, »LTV-fähige Signaturen in PLOP DS«, Seite 130.*

Beispiele für PAdES-Optionslisten. Folgende Signaturoption (zusätzlich zu anderen relevanten Optionen wie *digitalid)* erstellt eine Signatur gemäß PAdES-Stufe E-BES (da es sich hierbei um den Standardwert handelt, kann diese Option auch weggelassen werden):

```
sigtype=cades
```

Der folgende Teil einer Signatur-Optionsliste erzeugt eine Signatur gemäß PAdES-Signaturstufe E-EPES (mit Hilfe eines fiktiven Signaturrichtlinien-Identifikators):

```
policy={oid=2.16.276.1.89.1.1.1.1.3 commitmenttype=origin}
```

Der folgende Teil einer Signatur-Optionsliste erzeugt eine Signatur gemäß PAdES-Signaturstufe B-T:

```
timestamp={critical source={url={http://timestamp.acme.com/tsa}}}
```

Der folgende Teil einer Signatur-Optionsliste erzeugt eine Signatur gemäß PAdES-Signaturstufe B-LT:

```
timestamp={critical source={url={http://timestamp.acme.com/tsa}}} ltv=full
```

Der folgende Teil einer Signatur-Optionsliste erzeugt eine Signatur mit eingebettetem Zeitstempel mit einem zusätzlichen Archiv-Zeitstempel gemäß PAdES-Signaturstufe B-LTA:

```
ltv=full timestamp={critical source={url={http://timestamp.acme.com/tsa}}} ↵
doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

Mit dem folgenden Teil einer Signatur-Optionsliste kann eine vorhandene PAdES-Signatur der Signaturstufe B-LT mit einem zusätzlichen Archiv-Zeitstempel gemäß PAdES-Signaturstufe B-LTA erweitert werden:

```
ltv=full doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

8 API-Referenz für die PLOP- und PLOP DS-Bibliothek

8.1 Optionslisten

Optionslisten sind eine leistungsstarke wie einfache Methode zur Steuerung von PLOP-Operationen. Statt eine Vielzahl von einzelnen Funktionsparametern zu verlangen, unterstützen viele API-Methoden Optionslisten *(optlists)*. Dabei handelt es sich um Strings, die eine beliebige Anzahl von Optionen enthalten können. Optionslisten unterstützen verschiedene Datentypen und zusammengesetzte Datenstrukturen wie Arrays. In den meisten Sprachbindungen lassen sich Optionslisten problemlos durch Konkatenieren der erforderlichen Schlüsselwörter und Werte bilden. C-Programmierer können zur Erstellung von Optionslisten die Funktion *sprintf( )* nutzen. Eine Optionsliste ist ein String mit einem oder mehreren Paaren im Format

```
name value(s)
```

Namen und Werte sowie die Paare selbst können durch beliebigen Leerraum, z.B. Leerzeichen, Zeilenumbruch, Tabulatorzeichen oder Newline getrennt werden. Der Wert kann aus einer Liste von mehreren Werten bestehen. Zwischen Name und Wert können Sie auch ein Gleichheitszeichen '=' verwenden:

```
name=value
```

Einfache Werte. Einfache Werte können einen der folgenden Datentypen verwenden:

- Boolean *true* oder *false*; wird bei einer Option Booleschen Typs kein Wert angegeben, wird von *true* ausgegangen. Als abkürzende Schreibweise kann *noname* statt *name=false* verwendet werden.
- Strings: Strings, die Leerzeichen oder Gleichheitszeichen '=' enthalten, müssen mit *{* und *}* geklammert werden. Ein leerer String kann durch ein Klammernpaar *{ }* dargestellt werden. Vor den Zeichen *{, }* und \ muss ein zusätzlicher Backslash \ stehen, wenn sie zum String gehören sollen.
- Text-Strings sind eine spezielle Art von Strings für bestimmte Optionen. Sie können im Gegensatz zu Optionen vom Typ String nicht nur aus ASCII-Zeichen bestehen, sondern auch aus Unicode-Werten. In Unicode-fähigen Sprachbindungen können Sie einfach beliebige Werte für diese Optionen angeben (siehe »Unicode-Unterstützung in Sprachbindungen«, Seite 141). In nicht Unicode-fähigen Sprachbindungen müssen Sie dem Text-String einen UTF-8-BOM voranstellen, falls der String als UTF-8 interpretiert werden soll. Ist kein UTF-8 BOM vorhanden, werden Text-Strings im Encoding *auto* dargestellt, d.h. unter Windows in der aktuellen Codepage, unter zSeries in *ebcdic* und unter Unix und OS X in *iso8859-1*.
- Schlüsselwort: Schlüsselwort aus einer vordefinierte Liste
- Floats und Integers: dezimale Gleitkomma- oder Ganzzahlen; zur Trennung von Vor- und Nachkommastellen sind Punkt und Komma zulässig.
- Handle: verschiedene Objekt-Handles, zum Beispiel Dokument- oder Seiten-Handles. Technisch gesehen handelt es sich um Integer-Werte.

Je nach Typ und Interpretation einer Option können zusätzliche Einschränkungen bestehen. Integer- oder Float-Optionen können auf einen bestimmten Wertebereich beschränkt sein; Handles müssen für den zugehörigen Objekttyp gelten usw. Optionsbedingungen sind bei den entsprechenden Funktionsbeschreibungen dokumentiert. Einige Beispiele für einfache Werte (die erste Zeile zeigt einen String mit einem Leerzeichen):

```
password={secret string}
linearize=true
```

Listenwerte. Listenwerte bestehen aus mehreren Werten, die einfache Werte oder wiederum Listenwerte sein können. Listen werden mit *{* und *}* geklammert. Beispiel für einen Listenwert:

```
permissions={ noprint nocopy }
```

Hinweis *Der Backslash \ erfordert in vielen Programmiersprachen eine gesonderte Behandlung.*

Nicht eingegrenzte String-Werte. In folgenden Situationen können die tatsächlichen Zeichen in einem Optionswert mit den Syntaxzeichen von Optionslisten in Konflikt stehen:

- Kennwörter oder Dateinamen können nicht ausgeglichene Klammern, Backslashes und andere Sonderzeichen enthalten
- Japanische SJIS-Dateinamen in Optionslisten (nur sinnvoll in nicht Unicode-fähigen Sprachbindungen)

Um beliebigen Text oder Binärdaten zu übergeben, die nicht mit den Syntaxelementen von Optionslisten kollidieren, lassen sich nicht eingeschlossene Optionswerte zusammen mit einer Längenangabe in folgenden Syntaxvarianten angeben:

```
key[n]=Wert
key[n]={Wert}
```

Die Dezimalzahl *n* hat folgende Bedeutung:

- in Unicode-fähigen Sprachbindungen: die Anzahl der UTF-16-Code-Einheiten
- in nicht Unicode-fähigen Sprachbindungen: die Anzahl von Bytes, aus denen der String besteht

Die geschweiften Klammern um den String-Wert sind optional, aber dringend empfohlen. Sie sind für Strings beginnend mit einem Leerzeichen oder anderen Trennzeichen erforderlich. Geschweifte Klammern, Trennzeichen und Backslashes im String-Wert werden als solche interpretiert.

Das folgende Beispiel zeigt ein Kennwort bestehend aus 7 Zeichen mit Leerzeichen und Klammern. Der gesamte String ist von geschweiften Klammern umschlossen, die nicht Teil des Optionswerts sind:

```
password[7]={ ab}c d}
```

Rechteck. Ein Rechteck besteht aus einer Liste von vier Float-Werten, die die *x*- und *y*-Koordinaten der linken unteren und der rechten oberen Ecke des Rechtecks festlegen. Die Koordinaten werden im Standardkoordinatensystem von PDF interpretiert, das seinen Ursprung in der linken unteren Ecke der Seite hat und auf der Einheit Punkt basiert.

Zum Beispiel:

```
rect={ 100 100 200 150}
```

Zur automatischen Größenberechnung ohne Verzerrung kann das Schlüsselwort *adapt* verwendet werden, siehe Abschnitt 7.3.1, »Visualisieren von Signaturen mit Grafik oder Logo«, Seite 105.

Unicode-Unterstützung in Sprachbindungen. Wenn eine Programmiersprache oder Entwicklungsumgebung intern Unicode-Strings unterstützt, nennen wir die Sprachbindung Unicode-fähig. Folgende Sprachbindungen sind Unicode-fähig:

- C++
- COM
- .NET
- Java
- Objective-C
- Python
- RPG

In diesen Umgebungen ist die Stringbehandlung einfach: Alle Strings werden automatisch als Unicode-Strings im Format UTF-16 übergeben. Vom Client übergebene Unicode-Strings werden von den Sprach-Wrappern korrekt verarbeitet, die automatisch auch bestimmte Optionen setzen.

Folgende Sprachbindungen sind nicht Unicode-fähig:

- C (kein integrierter String-Datentyp verfügbar)
- Perl
- PHP
- Ruby

Für nicht Unicode-fähige Sprachbindungen wird UTF-8 empfohlen. Bei einigen API-Elementen wird zwischen Unicode-fähigen und nicht Unicode-fähigen Sprachbindungen unterschieden. Diese Unterschiede werden in den jeweiligen API-Beschreibungen in diesem Kapitel erwähnt.

Mit der Konvertierungsfunktionen *PLOP_convert _to_unicode( )* lassen sich Strings zwischen den Formaten UTF-8, UTF-16- und UTF-32 oder von beliebigen Encodings nach Unicode mit einem optionalen BOM konvertieren.

8.2 Allgemeine Funktionen

C ***PLOP*PLOP_new(void)***

Erzeugt einen neuen PLOP-Kontext.

Rückgabe Handle auf den neuen Kontext oder NULL, falls nicht genügend Speicherplatz verfügbar ist. Der Kontext muss an alle weiteren API-Funktionen übergeben werden.

Bindungen Nicht verfügbar in objektorientierten Sprachbindungen, bei denen die Funktion bei der Erzeugung eines neuen PLOP-Objekts automatisch aufgerufen wird.

Java ***void delete( )***

C# ***void Dispose( )***

C ***void PLOP_delete(PLOP *plop)***

Löscht einen PLOP-Kontext und gibt alle zugehörigen internen Ressourcen frei.

Details Alle in diesem Kontext geöffneten Dokumente werden geschlossen. Es gehört jedoch zu gutem Programmierstil, Dokumente explizit mit *PLOP_close_document( )* zu schließen, wenn sie nicht mehr benötigt werden.

Bindungen In C darf diese Funktion innerhalb einer *PLOP_TRY( )/PLOP_CATCH( )*-Klausel nicht aufgerufen werden.

In Java wird diese Methode durch die Finalizer-Methode von PLOP aufgerufen. Wir empfehlen jedoch, *delete( )* für ein zuverlässiges Cleanup sowie zur Fehlerbereinigung explizit aufzurufen.

In Perl, PHP und COM wird diese Funktion automatisch aufgerufen, sobald ein PLOP-Objekt gelöscht wird.

In .NET sollte *Dispose( )* am Ende der Verarbeitung aufgerufen werden, um »unmanaged« Ressourcen zu bereinigen.

C++ ***void create_pvf(wstring filename, const void *data, size_t size, wstring optlist)***

C# Java ***void create_pvf(String filename, byte[ ] data, String optlist)***

Perl PHP ***create_pvf(string filename, string data, string optlist)***

C ***void PLOP_create_pvf(PLOP *plop,***
const char *filename, int len, const void *data, size_t size, const char *optlist)

Erzeugt eine benannte virtuelle, schreibgeschützte Datei aus Daten im Speicher.

filename (Name-String) Der Name der virtuellen Datei. Dies ist ein beliebiger String, mit dem in weiteren PLOP-Aufrufen die virtuelle Datei referenziert werden kann.

len (Nur C-Sprachbindung) Länge von *filename* (in Bytes) für UTF-16-Strings. Ist *len=0*, muss ein null-terminierter String übergeben werden.

data Daten für die virtuelle Datei. In COM ist es ein Variantentyp mit Bytes, der den Inhalt der virtuellen Datei enthält. In C und C++ handelt es sich um einen Zeiger auf einen Speicherbereich. In Java ist es ein Byte-Array. In Perl und PHP ist es ein String.

size (Nur C- und C++-Sprachbindung) Länge des Speicherblocks mit den Daten in Bytes.

optlist Optionsliste gemäß Tabelle 8.1. Die folgende Option kann verwendet werden: *copy*.

Details Diese Funktion kann bei mehrfach verwendeten digitalen IDs oder XMP-Metadaten nützlich sein. Der Name der virtuellen Datei kann an alle API-Funktionen übergeben werden, die Eingabedateien verarbeiten. Manche Funktionen sperren die virtuelle Datei, solange die Daten verwendet werden. Virtuelle Dateien werden solange im Speicher gehalten, bis sie mit *PLOP_delete_pvf( )* explizit oder mit *PLOP_delete( )* automatisch gelöscht werden.

PVF-Dateien werden für jedes PLOP-Objekt getrennt gespeichert. Virtuelle Dateien lassen sich nicht von verschiedenen PLOP-Objekten gemeinsam nutzen. Arbeiten Threads mit verschiedenen PLOP-Objekten, müssen sie den PVF-Gebrauch nicht synchronisieren. Verweist *filename* auf eine bereits existierende virtuelle Datei, wird eine Exception ausgelöst. Diese Funktion überprüft nicht, ob *filename* bereits für eine auf der Festplatte liegende Datei verwendet wird.

Wird die Option *copy* nicht angegeben, darf der Aufrufer die übergebenen Daten erst nach dem erfolgreichen Aufruf von *PLOP_delete_pvf( )* ändern oder freigeben (löschen). Bei Nichtbeachtung dieser Regel droht ein Absturz.

Tabelle 8.1 Option für PLOP_create_pvf( )

Option	Beschreibung
copy	*(Boolean) Bei* `true` *erzeugt PLOP eine interne Kopie der übergebenen Daten. In diesem Fall kann der Aufrufer die übergebenen Daten unmittelbar nach dem Aufruf löschen. Standardwert:* `false` *für C und C++, aber* `true` *für alle anderen Sprachbindungen.*

C++ Java ***int delete_pvf(String filename)***

Perl PHP ***int delete_pvf(string filename)***

C ***int PLOP_delete_pvf(PLOP *plop, const char *filename, int len)***

Löscht eine benannte virtuelle Datei und gibt die zugehörigen Datenstrukturen.

filename (Name-String) Der Name der virtuellen Datei wie an *PLOP_create_pvf( )* übergeben.

len (Nur C-Sprachbindung) Länge von *filename* (in Bytes) für UTF-16-Strings. Ist *len=0,* muss ein null-terminierter String übergeben werden.

Rückgabe -1 (in PHP: 0), falls die zugehörige virtuelle Datei existiert und gesperrt ist, sonst 1.

Details Ist die Datei nicht gesperrt, werden die zu *filename* gehörigen Datenstrukturen sofort von PLOP gelöscht. Verweist *filename* nicht auf eine virtuelle Datei, kehrt die Funktion sofort zurück. Nach einem erfolgreichen Aufruf kann *filename* wieder verwendet werden. Virtuelle Dateien werden durch *PLOP_delete( )* automatisch gelöscht.

Das genaue Verhalten hängt davon ab, ob das zugehörige *PLOP_create_pvf( )* mit der Option *copy* aufgerufen wurde: Ist dies der Fall, werden sowohl die administrativen Datenstrukturen der Datei als auch die eigentlichen Daten freigegeben; andernfalls obliegt die Freigabe des Inhalts dem Client.

C++ Java **double info_pvf(String filename, String keyword)**

Perl PHP **float info_pvf(string filename, string keyword)**

C **double PLOP_info_pvf(PDF *p, const char *filename, int len, const char *keyword)**

Liefert Eigenschaften einer virtuellen Datei oder des PDFlib Virtual File System (PVF).

filename (Name-String) Der Name der virtuellen Datei. Bei *keyword=filecount* kann der Dateiname leer sein.

len (Nur C-Sprachbindung) Länge von *filename* (in Bytes) für UTF-16-Strings. Ist *len=0*, muss ein null-terminierter String übergeben werden.

keyword Schlüsselwort gemäß Tabelle 8.2.

Tabelle 8.2 Schlüsselwörter für PLOP_info_pvf()

Schlüsselwort	Beschreibung
exists	*Gibt 1 aus, wenn die Datei im PDFlib Virtual File System existiert (und nicht gelöscht wurde), sonst 0.*
filecount	*Gesamtzahl der Dateien im PDFlib Virtual File System, die für das aktuelle PLOP-Objekt verwaltet werden. Die Option* `filename` *wird ignoriert.*
iscopy	*(Nur für bestehende virtuelle Dateien) Gibt 1 aus, wenn die Option* copy *bei Erstellung der angegebenen virtuellen Datei übergeben wurde, sonst 0.*
lockcount	*(Nur für bestehende virtuelle Dateien) Anzahl der Sperren für die angegebene virtuelle Datei, die von PLOP-Funktionen intern gesetzt wurden. Die Datei kann nur gelöscht werden, wenn der Zähler für die Sperren auf 0 steht.*
size	*(Nur für bestehende virtuelle Dateien) Gibt die Größe der entsprechenden virtuellen Datei in Bytes aus.*

Rückgabe Den Wert von *keyword* erwarteter Dateiparameter.

Details Liefert Eigenschaften einer virtuellen Datei oder des PDFlib Virtual File System (PVF). Die gewünschte Eigenschaft wird mit *keyword* angegeben.

8.3 Funktionen für die Eingabe

C++ Java ***int open_document(String filename, String optlist)***

Perl PHP ***int open_document(string filename, string optlist)***

C ***int PLOP_open_document(PLOP *plop, const char *filename, int len, const char *optlist)***

Öffnet ein eventuell geschütztes PDF-Dokument zur Verarbeitung.

filename Vollständiger Pfadname der zu verarbeitenden PDF-Datei. Die Datei wird mit Hilfe der *SearchPath*-Ressource gesucht.

In nicht Unicode-fähigen Sprachbindungen wird der Dateiname gemäß der Option *filenamehandling* nach UTF-8 konvertiert (sofern nicht *filenamehandling=unicode* oder der übergebene Dateiname mit einem UTF-8-BOM beginnt). Ist *len* von 0 verschieden (nur C-Sprachbindung), wird der Dateiname ungeachtet der Option *filenamehandling* von UTF-16 nach UTF-8 konvertiert. Wenn der Dateiname nicht konvertiert werden kann oder kein gültiges UTF-8 oder UTF-16 darstellt, wird ein Fehler zurückgegeben.

Unter Windows können UNC-Pfade oder Netzwerkfreigaben verwendet werden, sofern Sie die dazu erforderlichen Berechtigungen haben (was bei ASP nicht unbedingt der Fall ist).

len (Nur C-Sprachbindung) Länge von *filename* (in Bytes) für UTF-16-Strings. Ist *len=0*, muss ein null-terminierter String übergeben werden.

optlist Optionsliste (siehe Abschnitt 8.1, »Optionslisten«, Seite 139) gemäß Tabelle 8.3.

Rückgabe -1 (in PHP: 0) bei einem Fehler, sonst ein Dokument-Handle. Gibt die Funktion einen Fehler zurück, sollten Sie genauere Informationen zur Fehlerursache mit *PLOP_get_errmsg( )* abfragen.

Details Das Dokument-Handle kann für folgende Zwecke verwendet werden:

- Als Eingabedokument zur weiteren Verarbeitung mit *PLOP_create_document( );*
- Übergabe einer Seite als Signaturvisualisierung (Signaturoption *field* und Unteroption *visdoc);*
- Anzeige von Dokumentinformationen mit pCOS.

Ist das Dokument verschlüsselt, muss sein Benutzer- oder Master-Kennwort (oder bei Zertifikatsicherheit eine passende digitale ID) übergeben werden, sofern nicht die Option *requiredmode* angegeben wurde.

Tabelle 8.3 Optionen für PLOP_open_document( )*

Option	Beschreibung
digitalid	*(Optionsliste; nur für Eingabedokumente mit Zertifikatsicherheit) Digitale ID eines Empfängers, für den das Dokument verschlüsselt wird. Die digitale ID wird mit den Unteroptionen in Tabelle 8.9 angegeben. Bei* `engine=mscapi` *kann die Option* `digitalid` *fehlen oder leer sein, dann werden alle verfügbaren IDs im Standard-Zertifikatspeicher geprüft.*
engine	*(Schlüsselwort; nur für Eingabedokumente mit Zertifikatsicherheit) Kryptografische Engine zur Entschlüsselung des Dokuments (Standardwert:* `builtin`*):* **builtin** *Die interne kryptografische Engine wird verwendet; in der Option* `digitalid` *muss eine virtuelle Datei oder einer Datei auf der Festplatte angegeben werden.* **mscapi** *(Nur bei Windows) Microsoft Crypto API wird als kryptografische Engine verwendet. Die digitale ID kann im Zertifikatspeicher oder in einer Datei auf der Festplatte übergeben werden. Die Option* `digitalid` *kann fehlen oder leer sein.*
inmemory	*(Boolean; nur für PLOP_open_document( )) Bei* `true` *lädt PLOP die Datei vollständig in den Speicher und verarbeitet sie dort. Auf manchen Systemen (insbesondere z/OS) lässt sich die Leistung damit erheblich steigern, es ist jedoch mehr Speicher erforderlich. Bei* `false` *wird das Dokument stückweise von der Festplatte gelesen. Standardwert:* `false`
password	*(String; erforderlich für geschützte Dokumente außer mit* `requiredmode`*)* *Für kennwortgeschützte Dokumente: Benutzer- oder Master-Kennwort für das Dokument. Wie in Tabelle 5.2, Seite 66 dargestellt, kann das Benutzer-, Master- oder kein Kennwort für das Dokument erforderlich sein, je nachdem, welche Operation für das Dokument durchgeführt werden soll. Auf EBCDIC-Plattformen wird das Kennwort in EBCDIC-Encoding oder EBCDIC-UTF-8 erwartet. Bei* `update=true` *wird das selbe Kennwort auch als Master-Kennwort für das erzeugte Ausgabedokument verwendet.* *Wenn* `digitalid` *übergeben wird: Kennwort, Passphrase oder PIN für die digitale ID, die für Dokumente mit Zertifikatsicherheit erforderlich ist. Bei* `engine=builtin` *ist entweder* `password` *oder* `passwordfile` *erforderlich; andere Engines verwenden eventuell alternative Methoden. Auf EBCDIC-Plattformen wird das Kennwort in EBCDIC-Encoding erwartet.*

Tabelle 8.3 Optionen für PLOP_open_document( )*

Option	Beschreibung
passwordfile	*(String; für* `engine=builtin` *ist genau eine der Optionen* `password` *oder* `passwordfile` *erforderlich; andere Engines verwenden eventuell alternative Methoden) Die erste Zeile der Datei (ohne Zeilenendezeichen) wird als Kennwort, Passphrase oder PIN für die digitale ID verwendet. Auf EBCDIC-Plattformen wird der Inhalt der Kennwortdatei in EBCDIC-Encoding erwartet.*
repair	*(Schlüsselwort) Legt fest, wie beschädigte PDF-Eingabedokumente behandelt werden. Die Reparatur eines Dokuments benötigt zwar mehr Zeit als das normale Parsen, ermöglicht aber die Verarbeitung bestimmter beschädigter PDF-Dokumente. Beachten Sie, dass sich manche Dokumente trotz Reparatur nicht verarbeiten lassen (Standardwert:* `auto`*):* ***force*** *Es wird in jedem Fall versucht, das Dokument zu reparieren.* ***auto*** *Es wird nur versucht, das Dokument zu reparieren, wenn beim Öffnen Probleme auftreten.* ***none*** *Es wird nicht versucht, das Dokument zu reparieren. Bei Problemen mit dem PDF-Dokument, schlägt die Funktion fehl.*
requiredmode	*(Schlüsselwort) Minimaler pCOS-Modus* (`minimum`/`restricted`/`full`)*, der beim Öffnen akzeptiert wird. Der Funktionsaufruf scheitert, wenn der resultierende pCOS-Modus niedriger als der erforderliche Modus ist. Ist der Aufruf erfolgreich, so ist sichergestellt, dass der resultierende pCOS-Modus mindestens dem in dieser Option festgelegten Modus entspricht. Er kann jedoch auch höher sein; so ergibt sich zum Beispiel aus* `requiredmode=minimum` *für ein unverschlüsseltes Dokument der Modus* `full`*. Standardwert:* `full`
shrug	*(Boolean) Berechtigungseinschränkungen werden ignoriert (d.h. PDF-Verarbeitung ist zulässig), falls das Dokument sonst nur im eingeschränkten pCOS-Modus geöffnet werden könnte. Bei Kennwortschutz wurde das Dokument dabei mit einem Master-Kennwort verschlüsselt, es liegt jedoch höchstens das Benutzerkennwort vor. Bei Zertifikatsicherheit wurde dabei eine passende digitale Empfänger-ID übergeben, für die jedoch keine Master-Berechtigung vorliegt. Werden die Zugriffsbeschränkungen ignoriert, wird das pCOS-Pseudo-Objekt* `shrug` *auf* `true` *gesetzt. Standardwert:* `false`
xmppolicy	*(Schlüsselwort) Steuert die Behandlung von ungültigem XMP auf Dokumentebene im Eingabedokument. Da bei ungültigem XMP kein Konformitätseintrag gefunden werden kann, werden PDF/A-Dokumente nicht als solche behandelt. Unterstützte Schlüsselwörter (Standardwert:* `rejectinvalid`*):* ***rejectinvalid*** *Löst eine Exception für das ungültige XMP aus, die die Fehlermeldung der XMP-Analyse enthält, und stoppt die Verarbeitung.* ***ignoreinvalid*** *(Impliziert* `sacrifice={pdfa pdfua pdfvt pdfx}`*) Ungültiges XMP wird als nicht vorhanden interpretiert. Basierend auf den Dokument-Infofeldern wird Ausgabe-XMP erzeugt; es enthält die Fehlermeldung des XML-Parsers im Element* `<pdfx:Exception>`*.* ***remove*** *(Impliziert* `sacrifice={pdfa pdfua pdfvt pdfx}`*) Eingabe-XMP ignorieren, ungeachtet seiner Gültigkeit. Das Ausgabe-XMP wird neu erzeugt. Damit lassen sich unerwünschte Metadaten entfernen. Standard-Identifikatoren in XMP (z.B. für PDF/A) gehen verloren.*

C++ ***int open_document_callback(void *opaque, size_t filesize,***
size_t (*readproc)(void *opaque, void *buffer, size_t size),
int (*seekproc)(void *opaque, long offset), wstring optlist)

C ***int PLOP_open_document_callback(PLOP *plop, void *opaque, size_t filesize,***
size_t (*readproc)(void *opaque, void *buffer, size_t size),
int (*seekproc)(void *opaque, long offset), const char *optlist)

Öffnet ein (eventuell verschlüsseltes) PDF-Dokument mit Hilfe einer vom Benutzer übergebenen Funktion.

opaque Zeiger auf eine Benutzerdatenstruktur, der an *readproc* übergeben wird. PLOP verwendet weder den Zeiger noch die zugrunde liegenden Benutzerdaten.

filesize Größe des Dokuments in Bytes.

readproc Prozedur, die in der Lage ist, Byte-Blöcke des Dokuments der Länge *size* an den Speicherort *buffer* zu schreiben. Die Prozedur muss die Anzahl der tatsächlich gelieferten Bytes zurückgeben.

seekproc Prozedur, die die Leseposition des Dokuments auf Position *offset* einstellt. Die Prozedur muss im Fehlerfall -1 zurückgeben und sonst 0.

optlist Optionsliste (siehe Abschnitt 8.1, »Optionslisten«, Seite 139) gemäß Tabelle 8.3.

Rückgabe -1 (in PHP: 0) bei einem Fehler, sonst ein Dokument-Handle. Gibt die Funktion einen Fehler zurück, sollten Sie genauere Informationen zur Fehlerursache mit *PLOP_get_errmsg( )* abfragen.

Bindungen Nur in den C- und C++-Sprachbindungen verfügbar.

C++ Java ***void close_document(int doc, String optlist)***

Perl PHP ***close_document(long doc, string optlist)***

C ***void PLOP_close_document(PLOP *plop, int doc, const char *optlist)***

Schließt das angegebene Dokument.

doc Gültiges Dokument-Handle, das mit *PLOP_open_document*( )* erzeugt wurde.

optlist Optionsliste (siehe Abschnitt 8.1, »Optionslisten«, Seite 139) gemäß Tabelle 8.4.

Details Diese Funktion sollte vor *PLOP_delete( )* für jedes mit *PLOP_open_document*( )* geöffnete Dokument aufgerufen werden. Diese Funktion schließt das Dokument, das mit dem übergebenen Handle verknüpft ist, und gibt alle zugehörigen Ressourcen frei.

Tabelle 8.4 Option für PLOP_close_document( )

Option	Beschreibung
lastinthread	*(Boolean) Diese Option sollte nach der Verarbeitung des letzten Dokuments im aktuellen Thread auf* `true` *gesetzt werden, um Speicherlecks zu vermeiden. Nach* `lastinthread=true` *sollte PLOP_create_document( ) nicht noch einmal für das selbe PLOP-Objekt aufgerufen werden. Standardwert:* `false`

8.4 Funktionen für die Ausgabe

C++ Java ***int create_document(String filename, String optlist)***

Perl PHP ***int create_document(string filename, string optlist)***

C ***int PLOP_create_document(PLOP *plop, const char *filename, int len, const char *optlist)***

Erzeugt ein PDF-Ausgabedokument im Speicher oder in einer Datei auf der Festplatte.

filename (Name-String) Name der generierten Ausgabedatei, der vom Namen der an *PLOP_open_document( )* übergebenen Eingabedatei verschieden sein muss. Bei einem leeren String wird die Ausgabe im Arbeitsspeicher erzeugt und kann später mit *PLOP_get_buffer( )* abgeholt werden.

In nicht Unicode-fähigen Sprachbindungen werden Dateinamen mit *len=0* in der aktuellen System-Codepage interpretiert, sofern ihnen nicht ein UTF-8-BOM vorausgeht, in welchem Fall sie als UTF-8 oder EBCDIC-UTF-8 interpretiert werden.

len (Nur C-Sprachbindung) Länge von *filename* (in Bytes) für UTF-16-Strings. Ist *len=0,* muss ein null-terminierter String übergeben werden.

optlist Optionsliste (siehe Abschnitt 8.1, »Optionslisten«, Seite 139) gemäß Tabelle 8.5.

Rückgabe -1 (in PHP: 0) bei einem Fehler, sonst ein Dokument-Handle. Gibt die Funktion einen Fehler zurück, sollten Sie genauere Informationen zur Fehlerursache mit *PLOP_get_errmsg( )* abfragen.

Bei der Erzeugung einer digitalen Signatur schlägt der Funktionsaufruf in folgenden Fällen fehl:

- es konnte kein Zeitstempel erstellt werden und die Option *critical* ist gesetzt;
- die Signaturkette wird erneut auf Gültigkeit geprüft und ein Zertifikat ist inzwischen abgelaufen oder wurde gesperrt;
- ein zur Visualisierung einer Signatur übergebenes Dokument passt nicht auf die Seite;
- das Eingabedokument ist beschädigt und die Signatur wird im Aktualisierungsmodus erstellt.

Wurde *PLOP_add_recipient( )* ein- oder mehrfach aufgerufen, war aber nicht erfolgreich (d.h. alle Empfängerzertifikate wurden abgelehnt), schlägt *PLOP_create_document( )* fehl. Damit kann das versehentliche Erzeugen nicht verschlüsselter Ausgabe vermieden werden.

Details Vor dem Aufruf dieser Funktion muss *PLOP_open_document*( )* aufgerufen werden. Das zu verarbeitende Dokument wird in der Option *input* übergeben. Zu den Bedingungen, die für Benutzer- und Master-Kennwort gelten, siehe Abschnitt 5.2, »Kennwortschutz für Dokumente mit PLOP einrichten«, Seite 65.

Im Signaturmodus prüft *PLOP_create_document( )* die Signaturkette eventuell erneut auf Gültigkeit, z.B. weil eine OCSP-Antwort seit ihrer Anforderung abgelaufen ist.

Bei Zertifikatsicherheit, d.h. wenn *PLOP_add_recipient( )* mindestens einmal mit einer nicht leeren Optionsliste *certificate* aufgerufen wurde, prüft diese Funktion ob ein Empfängerzertifikat abgelaufen ist. Wenn ja, schlägt die Funktion fehl.

Tabelle 8.5 Optionen für PLOP_create_document( )

Option	Beschreibung
docinfo	*(Liste von Paaren von Text-Strings) Legt die Dokument-Infofelder für das Ausgabedokument fest. Enthält das Dokument XMP-Metadaten auf Dokumentebene, werden die Standard-Dokument-Infofelder im XMP gespiegelt. Jedes Paar in der Optionsliste enthält Namen und Wert eines Felds. Folgende vordefinierte und benutzerdefinierte Schlüssel können übergeben werden (Standardwert: Dokument-Infofelder werden aus dem Eingabedokument kopiert):* **Subject** *Thema des Dokuments* **Title** *Titel des Dokuments* **Author** *Verfasser des Dokuments* **Keywords** *Stichwörter für den Inhalt des Dokuments* **Trapped** *Gibt an, ob die Datei Überfüllungsinformationen enthält. Erlaubt sind die Werte* `True`, `False` *und* `Unknown`. *Bei PDF/X-Eingabe ist* `Unknown` *nur zulässig, wenn die Option* `sacrifice` *den Wert* `pdfx` *oder* `pdfvt` *enthält.* ***beliebiger Name außer Creator, CreationDate, Producer, ModDate, GTS_PDFXVersion, GTS_PDFXConformance, ISO_PDFEVersion*** *Benutzerdefinierter Feldname (darf keine Leerzeichen enthalten). PLOP unterstützt beliebig viele Felder. Der Name eines benutzerdefinierten Felds sollte nur einmal übergeben werden.*
encryption	*(Schlüsselwort; nur relevant bei Kennwortschutz und Zertifikatsicherheit) Verschlüsselungsalgorithmus zum Schutz des Ausgabedokuments.* *Folgende Schlüsselwörter werden für Kennwortschutz unterstützt, d.h. bei der Übergabe von* `masterpassword` *(Standardwert:* `algo11`*):* **algo4** *(In PDF 2.0 als veraltet deklariert) Verschlüsselung mit AES-128 gemäß Acrobat 7/8, d.h. pCOS-Algorithmus 4; dadurch erhöht sich die Version des Ausgabe-PDF auf PDF 1.6, falls erforderlich. Kennwörter dürfen nur aus dem Latin-1-Zeichensatz bestehen und sind auf 32 Zeichen beschränkt.* **algo11** *Verschlüsselung mit AES-256 gemäß Acrobat X/XI/DC, d.h. pCOS-Algorithmus 11; dadurch erhöht sich die Version des Ausgabe-PDF auf PDF 1.7ext8, falls erforderlich. Kennwörter dürfen Unicode-Zeichen enthalten und sind auf 127 UTF-8-Bytes beschränkt.* *Folgende Schlüsselwörter werden für Zertifikatsicherheit unterstützt, d.h. wenn PLOP_add_recipient( ) mindestens einmal mit der nicht leeren Optionsliste* `certificate` *erfolgreich aufgerufen wurde (Standardwert:* `algo10`*):* **algo6** *(In PDF 2.0 als veraltet deklariert) Verschlüsselung mit Zertifikatsicherheit in Kombination mit AES-128 gemäß Acrobat 7, d.h. pCOS-Algorithmus 6; dadurch erhöht sich die Version des Ausgabe-PDF auf PDF 1.6, falls erforderlich.* **algo10** *Verschlüsselung mit Zertifikatsicherheit in Kombination mit AES-156 gemäß Acrobat 9, d.h. pCOS-Algorithmus 10; dadurch erhöht sich die Version des Ausgabe-PDF auf PDF 1.7ext3, falls erforderlich.*
input	*(Dokument-Handle, das mit PLOP_open_document*( ) erzeugt wurde; erforderlich) Zu verarbeitendes Eingabedokument*
limitcheck	*Bei* `true` *wird der Höchstwert für die Anzahl von indirekten PDF-Objekten (8 388 607) in den Modi PDF/A-1/2/3 und PDF/X-4/5 erzwungen. Standardwert:* `true`
linearize	*(Boolean; kann nicht mit Signaturerstellung oder* `metadata` *kombiniert werden) Bei* `true` *wird das Ausgabedokument linearisiert. Unter MVS kann diese Option nicht mit der direkten Generierung im Speicher kombiniert werden (d.h. einem leeren Parameter* `filename`*). Standardwert:* `false`
masterpassword[1]	*(String; erzwingt* `update=false`*) Master-Kennwort, um das Dokument mit Kennwort zu verschlüsseln. Ist es leer oder fehlt es, wird kein Master-Kennwort angewendet. Standardwert: leer*

Tabelle 8.5 Optionen für PLOP_create_document( )

Option	Beschreibung
metadata	*(Optionsliste; kann nicht mit* linearize *kombiniert werden) Übergibt XMP-Metadaten für das Dokument. Einträge zur PDF/A- und PDF/X-Konformität sind im übergebenen XMP nicht zulässig. Unterstützte Unteroptionen:* ***filename*** *(Name-String; erforderlich) Name einer Datei mit wohlgeformten XMP-Metadaten im UTF-8-Format.* ***validate*** *(Schlüsselwort) Die übergebenen XMP-Metadaten werden gemäß dem Schlüsselwort validiert (Beachten Sie, dass PLOP die XMP-Metadaten im Eingabedokument nicht validiert):* ***none*** *Keine Validierung* ***xmp2004*** *Validierung gemäß der XMP-2004-Spezifikation* ***xmp2005*** *Validierung gemäß der XMP-2005-Spezifikation* ***pdfa1*** *Wie* xmp2004*, plus Testen der vordefinierten Eigenschaften und Schemas sowie Validierung der Extension-Schemas gemäß PDF/A-1* ***pdfa2*** *Wie* xmp2005*, plus Testen der vordefinierten Eigenschaften und Schemas sowie Validierung der Extension-Schemas gemäß PDF/A-2 und PDF/A-3 (die Anforderungen an Metadaten sind für beide Standards identisch)* *Standardwert:* pdfa1*, wenn die Eingabe PDF/A-1-konform ist und die Option* sacrifice *nicht* pdfa *enthält;* pdfa2*, wenn die Eingabe PDF/A-2- oder PDF/A-3-konform ist und die Option* sacrifice *nicht* pdfa *enthält; sonst* none
objectstreams	*(Schlüsselwort; wird bei* linearize=true *sowie im PDF/A-1- und PDF/X-1a/3-Modus auf* none *gesetzt) Erzeugt komprimierte Objekt-Streams, was die Größe der Ausgabedatei erheblich verringert (Standardwert:* all*):* ***all*** *Schreibt alle einfachen Objekte außer dem Dokumentinfo-Dictionary in komprimierte Objekt-Streams und erzeugt einen komprimierten Querverweis-Stream.* ***none*** *Erzeugt weder komprimierte Objekt-Streams noch einen komprimierten Querverweis-Stream.* ***xref*** *Erzeugt einen komprimierten Querverweis-Stream, aber keine anderen komprimierten Objekt-Streams.*
optimize	*(Schlüsselwort; wird bei* update=true *ignoriert) Anzuwendende Optimierungen (Standardwert:* none*):* ***all*** *Alle implementierten Optimierungen werden angewendet.* ***none*** *Keine Optimierung.*
permissions	*(Liste von Schlüsselwörtern; erfordert* masterpassword *oder Zertifikatsicherheit; nicht bei* update=true*) Liste der Zugriffsberechtigungen für das Dokument. Sie kann eine beliebige Anzahl der Schlüsselwörter* noprint*,* nomodify*,* nocopy*,* noannots*,* noassemble*,* noforms*,* noaccessible*,* nohiresprint *und* plainmetadata *enthalten (siehe Tabelle 5.3, Seite 66).* *Bei Zertifikatsicherheit ist nur das Schlüsselwort* plainmetadata *erlaubt; weitere Berechtigungen können in der Option* permissions *von PLOP_add_recipient( ) festgelegt werden. Standardwert: leer*
recordsize	*(Integer; nur für z/OS und USS) Record-Größe der Ausgabedatei. Standardwert: 0 (Ausgabe ohne feste Blockgröße)*
sacrifice	*(Liste von Schlüsselwörtern) Steuerung des Verhaltens bei Konflikten zwischen PDF-Dokumenteigenschaften und der gewünschten Aktion. Bei einem Konflikt erzeugt PLOP keine Ausgabe, sondern löst eine Exception aus. Sie können jedoch bestimmte Eigenschaften aufgeben, damit das Dokument noch verarbeitet werden kann. Die Schlüsselwörter aus Tabelle 8.6 werden unterstützt; sie werden ignoriert, sofern die Auslöser für Eingabe und Aktion nicht beide* true *sind. Standardwert: leere Liste, bei einem Konflikt wird eine Exception ausgelöst und keine Ausgabe erzeugt.*
tempdirname	*(String) Name eines Verzeichnisses für die temporären Dateien, die für die PLOP-interne Verarbeitung benötigt werden. Ist diese Option leer, werden temporäre Dateien im aktuellen Verzeichnis abgelegt. Ist die Option* tempfilename *vorhanden, so wird diese Option ignoriert. Standardwert: leer*

Tabelle 8.5 Optionen für PLOP_create_document( )

Option	Beschreibung
tempfilename	*(String; nur für MVS) Vollständiger Name einer temporären Datei, die für die PLOP-interne Verarbeitung erforderlich ist. Ist diese Option leer, generiert PLOP selbst einen eindeutigen Namen. Der Benutzer muss die temporäre Datei nach Ausführung von PLOP_close_document( ) selbst löschen. Wird diese Option übergeben, darf der Parameter* `filename` *nicht leer sein. Standardwert: leer*
user-password[1]	*(String; erfordert* `masterpassword`*) Benutzerkennwort, um das Dokument mit Kennwortschutz zu versehen. Ist es leer oder fehlt es, wird kein Benutzerkennwort angewendet. Standardwert: leer*

1. Für AES-256 (Algorithmus 11) können beliebige Unicode-Zeichen übergeben werden, für AES-128 (Algorithmus 4) dagegen nur Latin-1-Zeichen. Das übergebene Kennwort wird bei Algorithmus 11 auf 127 UTF-8-Bytes und bei Algorithmus 4 auf 32 Zeichen gekürzt. Auf EBCDIC-Plattformen wird das Kennwort in `ebcdic`*-Encoding oder EBCDIC-UTF-8 erwartet.*

Tabelle 8.6 Schlüsselwörter für die Option `sacrifice` *von PLOP_create_document( )*

Schlüsselwort	Beschreibung
encrypted-attachments	*(Auslöser für die Eingabe: das Dokument selbst ist nicht verschlüsselt, enthält aber einen oder mehrere verschlüsselte Dateianhänge; Auslöser für die Aktion: das passende Kennwort für den verschlüsselten Dateianhang wurde nicht mit der Option* `password` *übergeben). Wird dieses Schlüsselwort übergeben, werden verschlüsselte Dateianhänge entfernt, für die kein Kennwort vorliegt.* *Dokumente mit verschlüsselten Dateianhängen, für die kein passendes Kennwort vorliegt, können nicht verarbeitet werden, wenn mit* `update=true` *signiert wird.*
fields	*(Auslöser für die Eingabe: das Dokument enthält ein oder mehrere Formularfelder, die nicht für die Signatur bestimmt sind und für die* `NeedAppearances=true` *gesetzt ist; Auslöser für die Aktion: Signieren mit* `update=false`*). Wird dieses Schlüsselwort übergeben, werden alle Formularfelder einschließlich Signaturfeldern entfernt.*
pdfa	*(Auslöser für die Eingabe: das Dokument genügt einer der Konformitätsstufen von PDF/A-1, PDF/A-2 oder PDF/A-3; Auslöser für die Aktion: Erstellen einer Signatur mit der Option* `visdoc` *und einem inkompatiblen Dokument für die Visualisierung, eine der Optionen* `userpassword`*,* `masterpassword`*/*`permissions`*, Zertifikatsicherheitsmodus oder eine Signatur wird größer als 64K für PDF/A-1 und 32K für PDF/A-2/3) Wird dieses Schlüsselwort übergeben, kann PDF/A-Eingabe zwar verarbeitet werden, die Einträge zur PDF/A-Konformität werden jedoch entfernt.*
pdfua	*(Auslöser für die Eingabe: das Dokument ist PDF/UA-1-konform; Auslöser für die Aktion: Verschlüsselung mit der Option* `permissions` *und dem Schlüsselwort* `noaccessible`*) Wird dieses Schlüsselwort übergeben, kann Ausgabe erzeugt werden, die nicht mehr PDF/UA-konform ist; die Einträge zur PDF/UA-Konformität werden entfernt.*
pdfvt	*(Auslöser für die Eingabe: das Dokument ist PDF/VT-1- oder PDF/VT-2-konform; Auslöser für die Aktion: wie bei* `pdfx`*) Wird dieses Schlüsselwort übergeben, kann Ausgabe erzeugt werden, die nicht mehr PDF/VT-konform ist; die Einträge zur PDF/VT-Konformität werden entfernt.*

Tabelle 8.6 Schlüsselwörter für die Option `sacrifice` *von PLOP_create_document( )*

Schlüsselwort	Beschreibung
pdfx	*(Auslöser für die Eingabe: das Dokument ist PDF/X-1a- oder PDF/X-3/4/5-konform; Auslöser für die Aktion: Erstellen einer Signatur in Kombination mit* `Trapped=Unknown` *der Option* `docinfo` *oder mit der Unteroption* `visdoc` *der Option* `field` *oder eine der Optionen* `userpassword`, `masterpassword`/`permissions` *oder Zertifikatsicherheitsmodus) Wird dieses Schlüsselwort übergeben, kann PDF/X-Eingabe zwar verarbeitet werden, die Einträge zur PDF/X-Konformität werden jedoch entfernt. Ist das Dokument auch konform zu PDF/VT-1 oder PDF/VT-2, werden die Einträge zur PDF/VT-Konformität ebenfalls entfernt.*
signatures	*(Auslöser für die Eingabe: das Dokument enthält eine oder mehrere Signaturen; Auslöser für die Aktion: alle Aktionen, außer Signieren mit* `update=true`*; enthält das Dokument eine Zertifizierungssignatur, für die keine Änderungen zulässig ist, wird das Signieren mit* `update=true` *ebenfalls ausgelöst). Wird dieses Schlüsselwort übergeben und sind die Auslöser für Eingabe und Aktion beide erfüllt, werden vorhandene Signaturen gelöscht (das heißt Signaturwerte werden entfernt, nicht aber die entsprechenden Formularfelder), um Ausgabe mit ungültigen Signaturen zu vermeiden. Wenn die Auslöser für Eingabe und Aktion beide erfüllt sind und* `sacrifice={signatures}` *nicht übergeben wurde, schlägt PLOP_create_document( ) fehl.*

C++ ***const char *get_buffer(long *size)***

C# Java ***byte[ ] get_buffer( )***

Perl PHP ***string get_buffer( )***

C ***const char *PLOP_get_buffer(PLOP *plop, long *size)***

Holt den Pufferinhalt des Ausgabedokuments teilweise oder vollständig aus dem Arbeitsspeicher.

size Nur in der C-Sprachbindung erforderlich. Zeiger auf einen Speicherplatz, an dem die Länge des zurückgegebenen Puffers abgelegt wird.

Rückgabe Puffer mit Ausgabedaten. In COM ist dies ein Varianten-Array vorzeichenloser Bytes. Bei JavaScript mit COM ist es nicht möglich, die Länge des zurückgegebenen Varianten-Arrays abzufragen (bei anderen Sprachen mit COM ist dies aber möglich). Der Client muss den Pufferinhalt erst vollständig verarbeiten, bevor die nächste PLOP-Funktion aufgerufen wird.

Details Mit dieser Funktion kann PDF-Ausgabe nur abgeholt werden, wenn durch die Übergabe eines leeren Dateinamens an *PLOP_create_document( )* die Erzeugung im Arbeitsspeicher angefordert wurde (andernfalls wird die Ausgabe in eine Datei geschrieben). *PLOP_get_buffer( )* muss vor dem Aufruf von *PLOP_close_document( )* aufgerufen werden.

8.5 Zertifikatsicherheit

C++ Java ***int add_recipient(String optlist)***
Perl PHP ***int add_recipient(string optlist)***
C ***int PLOP_add_recipient(PLOP *plop, const char *optlist)***

Hinzufügen eines Empfängerzertifikats zum Schutz des Ausgabedokuments.

optlist Optionsliste (siehe Abschnitt 8.1, »Optionslisten«, Seite 139) mit Empfängerinformationen gemäß Tabelle 8.7:
certificate, *conformance*, *engine*, *oaephash*, *rsapadding*, *permissions*

Rückgabe -1 (in PHP: 0) im Fehlerfall, sonst 1. Gibt die Funktion einen Fehler zurück, sollten Sie genauere Informationen zur Fehlerursache mit *PLOP_get_errmsg( )* abfragen. Diese Funktion schlägt fehl, falls das Zertifikat nicht gefunden oder nicht zur Verschlüsselung von PDF-Dokumenten verwendet werden kann.

Details Wird diese Funktion mindestens einmal mit einer nicht leeren Option *certificate* aufgerufen, werden die Ausgabedokumente für alle angegebenen Empfängerzertifikate verschlüsselt. Zertifikate können in einer Datei auf der Festplatte oder in einer virtuellen Datei oder im Zertifikatspeicher von Windows übergeben werden. Diese Funktion kann beliebig oft aufgerufen werden, um die Empfängerliste für das Dokument aufzubauen. Für die meisten Anwendungsfälle ist es empfehlenswert, das Zertifikat des Dokumentverfassers in die Empfängerliste aufzunehmen, da dieser sonst nicht in der Lage sein wird, das geschützte Dokument zu öffnen.

Das übergebene Zertifikat muss die in »Anforderungen an Empfängerzertifikate«, Seite 82 beschriebenen Bedingungen erfüllen.

Eine beliebige Anzahl von Dokumenten kann mit mehrfachen Aufrufen von *PLOP_create_document( )* für den selben Empfänger verschlüsselt werden. Wird *PLOP_add_recipient( )* jedoch nach *PLOP_create_document( )* erneut aufgerufen, wird zuerst die Empfängerliste auf eine leere Liste zurückgesetzt, damit eine neue Empfängerliste erstellt werden kann.

Diese Funktion erzwingt den Wert *update=false* der Signatur-Unteroption, da die Empfängerliste im Update-Modus nicht verändert werden kann.

Tabelle 8.7 Optionen für PLOP_add_recipient( )

Option	Beschreibung
certificate	*(Optionsliste; erforderlich) Angabe eines Empfängerzertifikats zur Verschlüsselung des Dokuments. Das Zertifikat wird mit den Unteroptionen aus Tabelle 8.9 angegeben.* *Eine leere Liste deaktiviert Zertifikatsicherheit. Dies kann nützlich sein, um zwischen Zertifikatsicherheit und anderer Verarbeitung zu wechseln.*

Tabelle 8.7 Optionen für PLOP_add_recipient()

Option	Beschreibung
conformance	*(Schlüsselwort) Konformität der erzeugten Verschlüsselungsinformation (Standardwert:* `acrobat`*):*
	acrobat *Das Dokument kann mit Acrobat geöffnet werden (für die benötigten Acrobat-Versionen siehe Tabelle 6.1, Seite 79). Falls das Empfängerzertifikat einen Algorithmus verwendet, der von Acrobat nicht unterstützt wird, schlägt die Funktion fehl.*
	extended *Akzeptiert das Empfängerzertifikat, selbst wenn es einen der unten angegebenen Algorithmen benutzt. Eventuell lässt sich das Dokument mit Acrobat nicht öffnen:* *RSA-Zertifikate, bei denen die Schlüssellänge kein Vielfaches von 8 beträgt;* *ECC-Zertifikate mit einer Brainpool-Kurve (RFC 5639);* *Verschlüsselung mit der Option* `rsapadding=oaep`*.*
engine	*(Schlüsselwort) Kryptografische Engine zum Auffinden des Empfängerzertifikats. Standardwert:* `builtin`*:*
	builtin *Die interne kryptografische Engine wird verwendet; Zertifikate müssen in einer virtuellen Datei oder einer Datei auf der Festplatte übergeben werden.*
	mscapi *(Nur für Windows) Verwendung von Microsoft Crypto API als kryptografische Engine; Zertifikate können im Zertifikatspeicher oder in einer Datei auf der Festplatte übergeben werden.*
oaephash	*(Schlüsselwort; nur relevant für* `rsapadding=oaep`*) Hash-Funktion für die Verwendung in OAEP (Standardwert:* `sha256`*):* `sha1`*,* `sha256`*,* `sha384` *oder* `sha512`
rsapadding	*(Schlüsselwort) Padding-Methode für den Schlüsseltransport mit RSA (Standardwert:* `pkcs#1`*):*
	pkc#1 *RSA-Padding gemäß PKCS#1 v1.5. Diese Methode wird in allen Acrobat-Versionen unterstützt.*
	oaep *OAEP (Optimal Asymmetric Encryption Padding) gemäß RFC 3447. Diese Methode bietet Sicherheitsvorteile. Da OAEP nicht in Acrobat XI/DC unterstützt wird, erfordert dieses Schlüsselwort die Option* `conformance=extended`*.*
permissions	*(Liste von Schlüsselwörtern) Liste der Berechtigungseinschränkungen für den Empfänger. Für mehrere Empfänger lassen sich unterschiedliche Berechtigungen festlegen. Die Liste enthält eine beliebige Anzahl der Berechtigungsschlüsselwörter* `noprint`*,* `nomodify`*,* `nocopy`*,* `noannots`*,* `noassemble`*,* `noforms`*,* `noaccessible`*,* `nohiresprint` *gemäß Tabelle 5.3. Das folgende zusätzliche Schlüsselwort kann verwendet werden:*
	nomaster *Drucken, Bearbeiten und Inhaltsextraktion gemäß den angegebenen Berechtigungsschlüsselwörtern beschränken sowie Änderungen der Dokument-Sicherheitseinstellungen verhindern. Wird weder dieses noch eins der anderen Berechtigungsschlüsselwörter oben übergeben, hat der Empfänger die vollen Rechte an dem Dokument. Das Setzen der anderen Berechtigungsschlüsselwörter oben impliziert* `nomaster`*.*
	Das Schlüsselwort `plainmetadata` *kann hier nicht verwendet werden, sondern muss an PLOP_create_document() übergeben werden, da es nicht für einzelne Empfänger angewendet werden kann.*
	Eine leere Liste bedeutet, dass es für den Empfänger keine Berechtigungseinschränkungen gibt. Standardwert: leere Liste

8.6 Digitale Signaturen

Hinweis *Die Funktionalität für digitale Signaturen steht nur im Produkt PLOP DS zur Verfügung.*

C++ Java ***int prepare_signature(String optlist)***

Perl PHP ***int prepare_signature(string optlist)***

C ***int PLOP_prepare_signature(PLOP *plop, const char *optlist)***

Signaturoptionen zur Verfügung stellen.

optlist Optionsliste mit Signaturoptionen gemäß Tabelle 8.8:

- Optionen für das Signaturzertifikat (digitale ID): *digitalid, password, passwordfile*
- Optionen für Informationen zum Signaturkontext: *contactinfo, location, policy, reason*
- Optionen für Zeitstempel: *doctimestamp, timestamp*
- Option für das Visualisieren von Signaturen: *field*
- Optionen für Prüfinformationen: *certfile, crl, crldir, crlfile, ocsp, rootcertdir, rootcertfile, validate*
- Optionen für Zertifizierungssignaturen: *certification, preventchanges*
- Optionen zur Steuerung von Details zur Signaturerstellung: *conformance, engine, ltv, signature, sigtype*
- Option zur Steuerung von Signaturdetails: *dss, rsaencoding, timestampsize, update*

Rückgabe -1 (in PHP: 0) im Fehlerfall, sonst 1. Gibt die Funktion einen Fehler zurück, sollten Sie genauere Informationen zur Fehlerursache mit *PLOP_get_errmsg( )* abfragen. Der Funktionsaufruf kann aus folgenden Gründen fehlschlagen:

- die digitale ID des Unterzeichners kann nicht gefunden werden oder es kann nicht auf den privaten Schlüssel zugegriffen werden, weil z.B. Kennwort oder PIN falsch sind;
- die Validierung schlägt fehl, weil z.B. keine gültige OCSP-Antwort oder CRL ermittelt werden konnte und die entsprechende Option *critical* gesetzt ist;
- die Voraussetzungen für *ltv=full* oder *validate=full* konnten nicht erfüllt werden.

Nach einem fehlgeschlagenen Aufruf von *PLOP_prepare_ signature( )* empfehlen wir, die Funktion nicht wieder mit den selben Optionen aufzurufen, da ein PKCS#11-Token deaktiviert werden könnte, wenn ein falsches Kennwort/PIN zu oft übergeben wird.

Details Mit dieser Funktion vorbereitete Signaturoptionen können verwendet werden, um eine beliebige Anzahl von Signaturen mit *PLOP_create_document( )* zu erstellen. Die übergebenen Signaturoptionen werden für alle Signaturen verwendet, die mit *PLOP_create_document( )* erzeugt werden, bis zum nächsten Aufruf von *PLOP_prepare_ signature( )* (mit anderen Signaturoptionen oder der Option *nosignature).*

Die Optionsliste zur Signaturvorbereitung kann zu nicht determinierten Zeitpunkten vor der Erzeugung einer Signatur erneut abgearbeitet werden. Insbesondere wenn eine CRL oder OCSP-Antwort nach der Erstellung mehrerer Signaturen abgelaufen ist, werden die Signaturoptionen erneut abgearbeitet, um die Sperrinformationen zu aktualisieren.

Diese Funktion beendet eine PKCS#11-Session, die eventuell durch einem früheren Aufruf noch aktiv ist. Wenn Sie eine PKCS#11-Session explizit beenden müssen (zum

Beispiel um anderen Threads Zugriff auf den Token zu gewähren), können Sie diese Funktion mit der Option *signature=false* aufrufen.

Tabelle 8.8 Optionen für PLOP_prepare_signature( )

Option	Beschreibung
certfile	*(String; nicht für* `engine=mscapi`*) Name einer Datei, die ein oder mehrere CA-Zwischenzertifikate in PEM-Kodierung enthält, die zur Validierung und Einbettung der vollständigen Zertifikatskette erforderlich sind.*
certification	*(Schlüsselwort) Erstellt eine Zertifizierungssignatur (Autorensignatur) vom angegebenen Typ. Von* `none` *verschiedene Werte erstellen eine Zertifizierungssignatur und sollten nur für die erste Signatur in einem Dokument verwendet werden (Standardwert:* `none`*):* **formfilling** *Zertifizierungssignatur: erlaubt das Ausfüllen von Formularfeldern und das Signieren (aber nur durch Anklicken eines Signaturfelds, nicht über das Acrobat-Menü). Das Hinzufügen von Seiten durch Kopieren von Seiten-Templates ist ebenfalls erlaubt (nicht jedoch das manuelle Hinzufügen von Seiten), diese Methode wird jedoch selten verwendet. Alle anderen Änderungen machen die Signatur ungültig.* **formsandannotations** *Zertifizierungssignatur: Ausfüllen von Formularfeldern, Signieren, Hinzufügen von Seiten sowie die Kommentarfunktionen (d.h. Anmerkungen erstellen, ändern und löschen) sind erlaubt; alle anderen Änderungen machen die Signatur ungültig.* **nochanges** *Zertifizierungssignatur: jede Änderung macht die Signatur ungültig.* **none** *Genehmigungssignatur: das Dokument ist nicht zertifiziert.*
conformance	*(Schlüsselwort) Konformität der erzeugten Signatur (Standardwert:* `acrobat`*):* **acrobat** *Die Signatur kann mit Acrobat validiert werden (für die benötigten Acrobat-Versionen siehe Tabelle 7.1, Seite 104). Die Funktion schlägt fehl, wenn ein Zertifikat oder Optionen verwendet werden, die nicht mit Acrobat kompatibel sind.* **extended** *Akzeptieren des Signaturzertifikats sowie von Optionen, auch wenn die resultierende Signatur nicht in Acrobat geprüft werden kann. Die folgenden Funktionen erfordern* `conformance=extended`*:* *ECC-Zertifikate mit einer der Brainpool-Kurven (RFC 5639)* *Signaturoption* `ocsp`*, wobei die Unteroption* `hash` *einen Wert ungleich* `sha1` *hat.*
contactinfo	*(Text-String; nur relevant für* `digitalid`*) Informationen des Unterzeichners, damit ein Empfänger den Unterzeichner zur Bestätigung der Signatur kontaktieren kann (z.B. eine Telefonnummer). Dies wird als skalierbare Lösung zur Feststellung der Vertrauenswürdigkeit jedoch nicht empfohlen. In Acrobat 8/9/X werden die Kontaktinformationen im Dialog* `Unterschriftseigenschaften` *im Register* `Unterzeichner` *angezeigt. In Acrobat XI/DC werden keine Kontaktinformationen angezeigt.*

Tabelle 8.8 Optionen für PLOP_prepare_signature( )

Option	Beschreibung
crl	*(Optionsliste oder Schlüsselwort; außer* `crl=none` *nur relevant für* `digitalid`; *nicht für* `engine=mscapi`*)* *Fordert eine Zertifikatsperrliste (CRL) für das Signaturzertifikat an und bettet sie in die Signatur oder einen DSS ein, falls keine gültige OCSP-Antwort verfügbar ist. Unterstützte Unteroptionen (Standardwert:* `{source={ } critical=false}`*, d.h. die* `CRLdp`*-Erweiterung in der digitalen ID wird verwendet, falls vorhanden):* **critical** *(Boolean) Bei* `true` *wird eine Signatur nur erzeugt, wenn eine gültige CRL für das Signaturzertifikat abgerufen werden konnte; andernfalls tritt ein Fehler auf und es wird keine Signatur erstellt. Bei* `false` *wird die Einbettung der CRL ignoriert, wenn keine gültige CRL abgerufen werden kann. Standardwert:* `true` **filename** *(String) Name einer Datei, die eine CRL für das Signaturzertifikat im DER-Kodierung enthält. Ist die Option* `filename` *vorhanden, wird die* `CRLdp`*-Erweiterung im Signaturzertifikat ignoriert.* **source** *(Netzwerk-Optionsliste) Optionsliste mit dem CRL Distribution Point des Signaturzertifikats. Die Protokolle* `http` *und* `https` *werden unterstützt. Die Unteroption* `url` *der Option* `source` *oder die Option* `source` *selbst können weggelassen werden. In diesem Fall wird die Erweiterung* `(CRLdp)` *der digitalen ID als Quelle verwendet.* *Alle Unteroptionen der Netzwerkoptionsliste* `source` *außer* `url` *und* `httpauthentication` *werden auch auf CRL-Anfragen für Zertifikate außer für Signaturzertifikate angewendet. Dies unterstützt den gleichen Anmeldeinformationen (z.B.* `username/password`*) in CRL-Aufrufen für alle beteiligten Zertifikate.* *Falls die Erweiterung* `CRLdp` *nicht in der digitalen ID vorhanden ist, muss genau eine der Optionen* `filename` *oder* `source` *übergeben werden.* *Bei* `crl=none` *werden keine CRLs über das Netzwerk abgefragt, selbst wenn die Erweiterung* `CRLdp` *vorhanden ist. Dies betrifft alle beteiligten Zertifikate, nicht nur das Signaturzertifikat.*
crldir	*(String; nicht bei* `engine=mscapi`*) Name eines Verzeichnisses mit CRLs in PEM-Kodierung , die zur Validierung der beteiligten Zertifikate erforderlich sind. Für Informationen zu Dateinamen siehe »Namenskonventionen für Zertifikate und CRL-Dateien«, Seite 179.*
crlfile	*(String; nicht für* `engine=mscapi`*) Name einer Datei, die ein oder mehrere CA-Zwischenzertifikate in PEM-Kodierung enthält, die zur Validierung und Einbettung der vollständigen Zertifikatskette erforderlich sind.*
digitalid	*(Optionsliste; erforderlich für Genehmigungs- und Zertifizierungssignaturen) Gibt die digitale ID des Unterzeichners mit Unteroptionen gemäß Tabelle 8.9 an. Welche Unteroptionen unterstützt werden, hängt von der ausgewählten Engine ab.*
doctimestamp	*(Optionsliste; nicht für* `engine=mscapi`*) Erzeugt einen Zeitstempel auf Dokumentebene von einer vertrauenswürdigen Time-Stamp Authority (unter Verwendung der* `builtin`*-Engine). Unterstützte Unteroptionen: siehe Option timestamp*
dss	*(Boolean; nicht für* `engine=mscapi`*) Bei* `true` *werden Zertifikate und zugehörige Sperrinformationen in einen Document Security Store (DSS) eingebettet (siehe Abschnitt 7.3.3, »Document Security Store (DSS)«, Seite 110). Andernfalls werden sie in die Signatur eingebettet. Prüfinformationen für Signaturen mit eingebettetem Zeitstempel und Zeitstempelsignaturen auf Dokumentebene werden unabhängig von dieser Option immer in einen DSS eingebettet. Standardwert:* `true` *für* `sigtype=cades` *sowie für Eingabedokumente mit einem bestehenden DSS; sonst* `false`

Tabelle 8.8 Optionen für PLOP_prepare_signature( )

Option	Beschreibung
engine	*(Schlüsselwort) Kryptografische Engine für die Signatur. Standardwert:* `builtin`:
	builtin *Die interne kryptografische Engine wird verwendet; die digitale ID muss in einer virtuellen Datei oder einer Datei auf der Festplatte übergeben werden.*
	mscapi *(Nur für Windows) Verwendung von Microsoft Crypto API als kryptografische Engine; die digitale ID kann im Zertifikatspeicher oder in einer Datei auf der Festplatte übergeben werden.*
	pkcs#11 *Die PKCS#11-Schnittstelle wird verwendet, um die digitale ID von einem kryptografischen Token zu laden. Der Name der zugehörigen PKCS#11-DLL/dynamischen Bibliothek für den Token muss in der Unteroption* `filename` *der Option* `digitalid` *übergeben werden.*
field	*(Optionsliste; nur relevant für* `digitalid`*) Koordinaten und Inhalt des Formularfelds für die Signatur gemäß den Unteroptionen in Tabelle 8.10. Standardwert: eine unsichtbare Signatur wird erstellt*
location	*(Text-String; nur relevant für* `digitalid`*) Physischer Ort oder Host-Name, wo die Signatur erzeugt wird*
ltv	*(Schlüsselwort; nicht für* `engine=mscapi`*) Gibt an, ob das signierte Dokument für die Langzeitvalidierung (LTV) vorbereitet wird (Standardwert:* `try`*):*
	full *(Impliziert* `validate=full`*) Einbetten der Prüfinformationen zur Langzeitvalidierung des signierten Dokuments. Der Status LTV erfordert meist eine der Optionen* `rootcertdir` *oder* `rootcertfile`*; die Optionen* `certfile`*,* `ocsp` *und* `crl` *für zusätzliche Informationen zu Zertifikaten und Sperrinformationen können ebenfalls erforderlich sein. Der Aufruf schlägt fehl, wenn ein benötigtes Zertifikat oder die Sperrinformationen nicht abgerufen werden können.*
	none *Prüfinformationen werden nicht eingebettet. Das signierte Dokument ist kleiner, aber nicht LTV-fähig.*
	try *Alle verfügbaren Prüfinformationen werden eingebettet. Abhängig von den verfügbaren Zertifikaten und den Sperrinformationen kann das signierte Dokument LTV-fähig sein oder nicht.*
ocsp	*(Optionsliste oder Schlüsselwort; nicht für* `engine=mscapi`*) Konfigurieren der OCSP-Verarbeitung mit Unteroptionen gemäß Tabelle 8.11. (Standardwert:* `{source={ } critical=false}`*, d.h. die AIA-Erweiterung in der digitalen ID wird verwendet, sofern vorhanden.* *Bei der Option* `ocsp=none` *werden keine OCSP-Antworten angefordert, selbst wenn die AIA-Erweiterung vorhanden ist. Dies betrifft alle beteiligten Zertifikate, nicht nur das Signaturzertifikat.*
password	*(String, der leer sein kann; für* `engine=builtin` *ist genau eine der Optionen* `password` *oder* `passwordfile` *erforderlich; andere Engines können alternative Methoden verwenden) Gibt Kennwort, Passphrase oder PIN für die digitale ID an. Bei* `engine=pkcs#11` *muss diese Option die PIN für das kryptografische Token enthalten, sofern die PIN nicht am Token selbst eingegeben werden muss (z.B. einem Smartcard-Reader mit Tastatur). Auf EBCDIC-Plattformen wird das Kennwort in EBCDIC-Encoding erwartet.*
passwordfile	*(String; für* `engine=builtin` *ist genau eine der Optionen* `password` *oder* `passwordfile` *erforderlich; andere Engines verwenden eventuell alternative Methoden) Die erste Zeile der Datei (ohne Zeilenendezeichen) wird als Kennwort, Passphrase oder PIN für die digitale ID verwendet. Auf EBCDIC-Plattformen wird der Inhalt der Kennwortdatei in EBCDIC-Encoding erwartet.*

Tabelle 8.8 Optionen für PLOP_prepare_signature( )

Option	Beschreibung
policy	*(Optionsliste; nur für* `sigtype=cades`*; nicht zulässig, wenn* `reason` *angegeben wird; erforderlich für PAdES E-EPES) Signaturrichtlinie zur Validierung der Signatur. Unterstützte Unteroptionen:* **commitmenttype** *(Schlüsselwort) Der zur Signatur gehörende Commitment-Typ im Rahmen der angegebenen Richtlinie. Unterstützte Schlüsselwörter (Standardwert:* `none`*):* **approval** *Der Unterzeichner hat den Inhalt der Nachricht genehmigt.* **creation** *Der Unterzeichner hat die Nachricht erzeugt (aber nicht unbedingt genehmigt oder gesendet).* **delivery** *Der vertrauenswürdige Service Provider hat eine Nachricht in einem lokalen Store hinterlegt, auf den der Empfänger der Nachricht zugreifen kann.* **none** *In der Signatur ist kein Commitment-Typ angegeben.* **origin** *Der Unterzeichner bestätigt, die Nachricht erzeugt, genehmigt und gesendet zu haben.* **receipt** *Der Unterzeichner bestätigt, den Inhalt der Nachricht erhalten zu haben.* **sender** *Der Unterzeichner hat die Nachricht gesendet (aber nicht unbedingt erzeugt).* **notice** *(Text-String) Lesbare Beschreibung der Signaturrichtlinie* **oid** *(String; erforderlich) Object ID der Signaturrichtlinie* **uri** *(String) URI der Signaturrichtlinie*
prevent-changes	*(Boolean; nur wenn* `certification` *von* `none` *verschieden ist) Bei* `true` *lassen sich die Änderungen, die durch die Option* `certification` *verboten sind (weil sie die Zertifizierungssignatur ungültig machen würden) in Acrobat nicht durchführen, d.h. die entsprechenden Werkzeuge sind in der Benutzeroberfläche deaktiviert. Standardwert:* `true`
reason	*(Text-String; nur relevant für* `digitalid`*; nicht zulässig mit* `policy`*) Grund für die Signierung des Dokuments*
rootcertdir	*(String; nicht für* `engine=mscapi`*) Name eines Verzeichnisses, das die vertrauenswürdige Stammzertifikate in PEM-Kodierung enthält, die zur Validierung der Zertifikatskette erforderlich sind. Für Dateinamenskonventionen siehe »Namenskonventionen für Zertifikate und CRL-Dateien«, Seite 179.*
rootcertfile	*(String; nicht für* `engine=mscapi`*) Name einer Datei, die ein oder mehrere vertrauenswürdige Stammzertifikate in PEM-Kodierung enthält, die zur Validierung der Zertifikatskette erforderlich sind. Aus Sicherheitsgründen wird die Datei nicht in* `searchpath` *gesucht.*
rsaencoding	*(Schlüsselwort; nur für* `engine=builtin`*) Encoding-Methode für RSA-Signaturen (Standardwert:* `pkcs#1`*):* **pkcs#1** *RSA-Encoding gemäß PKCS#1 v1.5. Diese Methode wird in allen Acrobat-Versionen unterstützt.* **pss** *RSA-Encoding gemäß PSS gemäß RFC 3447/RFC 8017, das Sicherheitsvorteile bietet. Mit dieser Methode erstellte Signaturen erfordern jedoch Acrobat XI/DC für die Validierung.*
signature	*(Boolean) Bei* `false` *wird keine Signatur erstellt. Dies kann nützlich sein, um zwischen dem Signieren und anderen Verarbeitungsschritten zu wechseln, selbst wenn in einem früheren Aufruf von PLOP_prepare_signature( ) Signaturoptionen übergeben wurden. Standardwert:* `true`
sigtype	*(Schlüsselwort; nur relevant für* `digitalid`*; nicht bei* `engine=mscapi`*) Signaturtyp (Standardwert:* `cades`*):* **cms** *CMS-basierte Signatur gemäß ISO 32000-1 und PAdES Teil 2 (ETSI TS 102 778-2)* **cades** *CAdES-basierte Signatur gemäß CAdES (ETSI TS 101 733) und RFC 5126. Dies ist eine Voraussetzung für PAdES Teil 3 und Teil 4.*

Tabelle 8.8 Optionen für PLOP_prepare_signature( )

Option	Beschreibung
timestamp	*(Optionsliste oder Schlüsselwort; nicht für* `engine=mscapi`*) Die Signatur enthält einen eingebetteten Zeitstempel einer vertrauenswürdigen Time-Stamp Authority (TSA). Unterstützte Unteroptionen (Standardwert:* `{source={ } critical=false}`*, d.h. die Erweiterung* `TimeStamp` *in der digitalen ID wird verwendet, sofern vorhanden):* **critical** *(Boolean; wird bei Zeitstempeln auf Dokumentebene auf* `true` *gesetzt) Bei* `true` *wird eine Signatur nur erstellt, wenn ein gültiger Zeitstempel erstellt werden kann; andernfalls tritt ein Fehler auf. Bei* `false` *wird der Zeitstempel ignoriert, wenn kein gültiger Zeitstempel erstellt werden kann. Standardwert:* `true` **hash** *(Schlüsselwort) Hash-Algorithmus für die Erzeugung der Zeitstempel-Anforderung. Der Algorithmus muss von der TSA unterstützt werden (Standardwert:* `sha256`*):* `sha1` *(nicht empfohlen),* `sha256`*,* `sha384` *oder* `sha512` **policy** *(String) OID der TSA-Richtlinie, mit der der Zeitstempel erzeugt werden muss. Unterstützt die TSA die angegebene Richtlinie nicht, kann kein Zeitstempel erstellt werden.* **source** *(Netzwerk-Optionsliste gemäß Tabelle 8.12) Optionliste, die die TSA beschreibt. Die Protokolle* `http` *und* `https` *werden unterstützt.* *Nur für eingebettete Zeitstempel, nicht jedoch für Zeitstempel auf Dokumentebene: die Unteroption* `url` *der Option* `source` *oder die Option* `source` *selbst können weggelassen werden. In diesem Fall wird die Erweiterung* `TimeStamp` *in der digitalen ID wird verwendet.* *Bei* `none` *wird kein Zeitstempel eingebettet, selbst wenn die* `TimeStamp`*-Erweiterung im Signaturzertifikat vorhanden ist.*
timestamp-size	*(Integer) Geschätzte Größe von Zeitstempel-Objekten zur Reservierung von Platz im CMS-Container für Zeitstempeln auf Dokumentebene und Signaturen mit eingebettetem Zeitstempel. Standardwert: 7168*
update	*(Boolean) Bei* `true` *werden die Signaturdaten als ein oder mehrere inkrementelle PDF-Updates an eine Kopie des Originaldokuments angehängt. Andernfalls wird die PDF-Objekthierarchie neu geschrieben, wodurch vorhandene Signaturen verloren gehen. Prüfinformationen für eingebettete Zeitstempel und Zeitstempel auf Dokumentebene werden unabhängig von dieser Option immer als Update angehängt. Für beschädigte Dokumente ist kein Update-Modus möglich.* *Standardwert:* `true`*, aber die Verschlüsselung der Ausgabe setzt den Wert der Option auf* `false` *(d.h. die Optionen* `masterpassword` *oder* `userpassword` *von PLOP_create_document( ) oder ein Aufruf von PLOP_add_recipient( ) mit einer nicht leeren Optionsliste* `certificate`*)*
validate	*(Schlüsselwort) Steuert die Validierung der beteiligten Zertifikate (Standardwert:* `full` *bei* `ltv=full`*, sonst* `formal`*):* **formal** *Folgendes wird geprüft:* *Kritische Erweiterungen, Schlüsselverwendung usw. wird geprüft;* *Eine OCSP-Antwort wird abgerufen, sofern angefordert und erfordert eine gültige Antwort mit dem Status* `good`*;* *CRL wird abgerufen, sofern angefordert und das Signaturzertifikat wird auf CRL geprüft; CRL-Daten werden geprüft;* **full** *Wie* `validate=formal` *sowie zusätzlich vollständige Validierung der Zertifikatskette. Dazu müssen alle vorliegenden Stamm- und Zwischenzertifikate verfügbar sein, sowie die OCSP- und CRL-Sperrinformationen für alle beteiligten Zertifikate (außer für die Stammzertifikate und OCSP-Responder mit der Erweiterung* `id-pkix-ocsp-nocheck`*).*

Tabelle 8.9 Unteroptionen der Option digitalid *von PLOP_prepare_signature( ) und PLOP_open_document( ) sowie die Option* certificate *von PLOP_add_recipient( )*

Option	Beschreibung
Unteroption für engine=builtin (falls der Aufruf über die Option digitalid ***oder*** certificate ***erfolgt):***	
filename[1]	*(String; erforderlich) Aufruf über die Option* digitalid*: Name einer Datei auf der Festplatte oder einer virtuellen Datei mit digitalen IDs im Format PKCS#12 oder PFX (für Hinweise zur Konvertierung siehe Anhang A, »Arbeiten mit Zertifikaten«).* *Aufruf über die Option* certificate*: Name einer Datei auf der Festplatte oder einer virtuellen Datei mit X.509-Zertifikaten in PEM- oder DER-Kodierung. Die Zertifikatdatei muss genau ein Zertifikat enthalten.*
Unteroption für engine=pkcs#11 (falls der Aufruf über die Option digitalid ***erfolgt):***	
externalhash	*(Boolean) Bei* true *wird der Dokument-Hash für die Signatur über die PKCS#11-Schnittstelle erzeugt (d.h. auf dem Token oder HSM), andernfalls in der integrierten Engine. Standardwert:* false
filename	*(String; erforderlich) Name der PKCS#11-DLL/dynamischen Bibliothek für das kryptografische Token, z.B. eine Smartcard. Es muss sich dabei um eine Datei auf der Festplatte handeln und darf keine PVF-Datei sein. Beispiel:* cryptoki.dll
issuer	*(String) Auswahl einer digitalen ID durch ihr Feld* issuer *(PKCS#11-Attribut CKA_ISSUER). Für eine Beschreibung des Abfrageformats siehe die Option* subject *unten.*
label	*(String) Auswahl einer digitalen ID durch ihr benutzerfreundliches Label (PKCS#11-Attribut CKA_ LABEL).*
serial	*(String) Auswahl einer digitalen ID durch ihre Seriennummer (PKCS#11-Attribut CKA_SERIAL_NUMBER). Die Seriennummer muss als dezimaler oder hexadezimaler String (mit dem Präfix* 0x*) übergeben werden, z.B.* serial=0x03A247B2
slotid	*(Positive Ganzzahl) Slot-Nummer für die Schnittstelle zum Token. Diese kann bei mehreren Slots zur direkten Auswahl eines Slots verwendet werden.*
subject	*(String) Auswahl einer digitalen ID durch ihr Feld* subject *(PKCS#11-Attribut CKA_ SUBJECT). Die Abfrage muss folgendem Format entsprechen:* /type0=value0/type1=value1/...*; Zeichen können durch einen Backslash (\) geschützt werden. Die Reihenfolge der Attribute ist signifikant. Enthält der Token mehr als eine digitale ID, können die Optionen* issuer*,* label *und* subject *zur Auswahl eines Zertifikats verwendet werden.* *Beispiel:* subject={/C=DE/L=Munich/O=PDFlib GmbH/CN=PLOP Demo Signer RSA-2048}
sticky	*(Boolean) Bei* true *bleibt die PKCS#11 DLL/dynamische Bibliothek bis zum Ende des Prozesses geladen. Dies kann Leistungsvorteile bieten und auch nützlich sein, um Probleme in der DLL/dynamischen Bibliothek, wie zum Beispiel bei Speicherlecks in ihrer Initialisierungsroutine zu umgehen. Es kann dann jedoch keine andere PKCS#11-DLL/dynamische Bibliothek im selben Prozess geladen werden. Sobald diese Option auf* true *gesetzt wurde, müssen weitere Aufrufe innerhalb des selben Prozesses ebenfalls* sticky=true *übergeben und die Unteroption* filename *wird ignoriert. Bei* false *wird die PKCS#11-Bibliothek im Aufruf von PLOP_delete( ) für das letzte in der Bibliothek verwendete PLOP-Objekt entladen.*
threadsafe	*(Boolean) Bei* true *muss die PKCS#11-Bibliothek thread-sichere Operationen unterstützen und wird in einem thread-sicheren Modus initialisiert. Unterstützt die PKCS#11-Bibliothek keine thread-sicheren Operationen, schlägt der Aufruf fehl. Bei* false *wird die PKCS#11-Bibliothek im single-threaded Modus initialisiert, was nur für Single-threaded-Anwendungen zulässig ist. Standardwert:* true

Tabelle 8.9 Unteroptionen der Option `digitalid` *von PLOP_prepare_signature( ) und PLOP_open_document( ) sowie die Option* `certificate` *von PLOP_add_recipient( )*

Option	Beschreibung
Unteroptionen für engine=mscapi (falls der Aufruf über die Option `digitalid` ***oder*** `certificate` ***erfolgt):***	
filename[1]	*(String; eins von* `filename` *oder* `store` *ist erforderlich, falls es von PLOP_prepare_signature( )*[2] *aufgerufen wird) Name einer Datei von der Festplatte oder einer virtuellen Datei mit digitalen IDs im Format PKCS#12 oder PFX (für Hinweise zur Konvertierung siehe Anhang A, »Arbeiten mit Zertifikaten«).*
storelocation	*(Schlüsselwort) Verzeichnis des Zertifikatspeichers (Standardwert:* `current_user`*):* `current_service`, `current_user`, `current_user_group_policy`, `local_machine`, `local_machine_enterprise`, `local_machine_group_policy`, `services`, `users` *Folgende Speicherorte können von einem anderen Computer geöffnet werden, indem der Option* `store` *der Computername vorangestellt wird (getrennt durch einen Backslash):* `local_machine`, `local_machine_group_policy`, `services`, `users`.
subject	*(String; erforderlich, falls von PLOP_prepare_signature( ) oder PLOP_add_recipient( ) aufgerufen und* `store` *angegeben wird; sonst ignoriert) Auswahl einer digitalen ID, wobei das Feld* `subject` *den übergebenen String enthält. Es enthält in der Regel den* `common name` *(CN) der digitalen ID. Diese Unteroption ist nicht erforderlich, wenn sie von PLOP_open_document( ) aufgerufen wird, da PLOP die entsprechende ID automatisch bestimmt.*
store	*(String; eine der Optionen* `filename` *oder* `store` *ist erforderlich, wenn sie von PLOP_prepare_signature( ) aufgerufen wird) Name des Zertifikatspeichers, z.B.* `My`, `Root`, `Trust`. *Bei* `storelocation=services` *oder* `storelocation=users` *muss dem Namen des Speichers der Service- oder Benutzername vorangestellt werden (getrennt durch einen Backslash). Standardwert:* `My`

1. Aus Sicherheitsgründen wird die Datei nicht in `searchpath` *gesucht, falls sie von der Option* `certificate` *aufgerufen wurde.*
2. Wurde in PLOP_prepare_signature( ) weder `filename` *noch* `store` *übergeben, werden alle verfügbaren IDs im Standardspeicher geprüft.*

Tabelle 8.10 Unteroptionen der Option `field` *von PLOP_prepare_signature( )*

Option	Beschreibung
fillexisting	*(Boolean; nur relevant, wenn ein oder mehrere Signaturfelder im Dokument vorkommen und die Option* `name` *nicht übergeben wird) Bei* `true` *wird das erste Signaturfeld im Eingabedokument zum Signieren verwendet. Bei* `false` *wird ein neues Signaturfeld mit einem eindeutigen Namen nach dem Muster* `Signature#` *erzeugt. Diese Option wird auf* `true` *gesetzt, wenn die Option* `visdoc` *im PDF/UA-Modus übergeben wird. Standardwert:* `false`
name	*(Text-String; darf nicht mit einem Punkt ».« enden) Name eines vorhandenen oder neuen Signaturfelds. Enthält das Dokument ein Signaturfeld mit diesem Namen, wird es für die Signatur verwendet (und* `page` *wird ignoriert), andernfalls wird das Feld erzeugt. Ist ein Feld mit diesem Namen zwar vorhanden, jedoch nicht vom Typ* `Unterschriftsfeld`*, tritt ein Fehler auf.* *Standardwert: ist kein Signaturfeld vorhanden, wird ein neues Feld mit dem Namen* `Signature1` *erzeugt. Andernfalls wird die Felderzeugung mit der Option* `fillexisting` *gesteuert.*
page	*(Positiver Integer; wird ignoriert, wenn ein vorhandenes Signaturfeld ausgefüllt wird) Nummer der Seite, auf der das Signaturfeld erzeugt wird. Die erste Seite hat die Nummer 1. Standardwert: 1*
position	*(Liste aus zwei Schlüsselwörtern) Relative Position der Visualisierungsseite innerhalb des Felds. Positioniert die Visualisierungsseite gemäß der übergebenen Schlüsselwörter und skaliert sie so, dass sie unter Beibehaltung ihrer Proportionen vollständig in das Rechteck passt. Das erste Schlüsselwort gibt die horizontale Position mit einem der Werte* `left`, `center` *oder* `right` *an; das zweite Schlüsselwort gibt die vertikale Position mit einem der Werte* `top`, `center` *oder* `bottom` *an. Sind die beiden Werte gleich, so kann ein Schlüsselwort weggelassen werden. Standardwert:* `{center}`

Tabelle 8.10 Unteroptionen der Option `field` *von PLOP_prepare_signature( )*

Option	Beschreibung
rect	*(Rechteck) Koordinaten der linken unteren und rechten oberen Ecke des Signaturfelds in PDF-Koordinaten (eine Einheit entspricht 1/72 Zoll, mit dem Ursprung in der linken unteren Ecke). Das angegebene Rechteck wird von der Visualisierungsseite vollständig ausgefüllt. Um eine Verzerrung zu vermeiden, kann statt ein oder zwei Koordinaten auch das Schlüsselwort* `adapt` *übergeben werden. In diesem Fall werden die fehlende(n) Koordinate(n) automatisch berechnet. Mindestens eine Ecke muss explizit angegeben werden. Das Rechteck darf nicht über die Seite hinausreichen. Für weitere Informationen zu den Optionen für die Objekteinpassung siehe »Position und Größe des Signaturfelds«, Seite 106. Ein leeres Rechteck mit vier Nullwerten bedeutet ein unsichtbares Feld.* *Standardwert: wird ein vorhandenes Feld verwendet, dient dessen Rechteck als Standard; sonst ein leeres Rechteck (d.h. die Signatur ist unsichtbar)*
tooltip	*(Nicht leerer Text-String) Tooltip-Text (Alternativtext) für ein Signaturfeld. Er kann von Screenreadern zur Verbesserung der Zugänglichkeit verwendet werden. Standardwert: keiner*
visdoc	*(Dokument-Handle, das mit PLOP_open_document( ) erzeugt wurde; nicht zulässig im PDF/X- oder PDF/VT-Modus; nur zulässig für ein nicht leeres Feldrechteck und in diesem Fall erforderlich) Dokument, aus dem eine Seite zur Visualisierung der Signatur auf der Seite verwendet wird. Im PDF/A-Modus muss das Visualisierungsdokument zur erzeugten Ausgabe kompatibel sein (siehe »PDF/A-Konformität«, Seite 107). Im PDF/UA-Modus muss das Eingabedokument ein geeignetes Signatur-Formularfeld enthalten (siehe »PDF/UA-Konformität«, Seite 109.*
vispage	*(Integer; nur relevant für* `visdoc`*) Seitenzahl im Dokument, die zur Visualisierung der Signatur verwendet wird (die erste Seite hat die Nummer 1). Standardwert: 1*

Tabelle 8.11 Unteroptionen der Option `ocsp` *von PLOP_prepare_signature( )*

critical	*(Boolean; nur relevant für* `digitalid`*) Bei* `true` *wird eine Signatur nur erstellt, wenn eine gültige OCSP-Antwort mit dem Status* `good` *für das Signaturzertifikat zurückgegeben wurde; andernfalls tritt ein Fehler auf und es wird keine Signatur erstellt. Bei* `false` *wird die Einbettung der OCSP-Antwort ignoriert, wenn keine gültige OCSP-Antwort abgerufen werden kann. Standardwert:* `true`
freshness	*(Integer) Maximaler Zeitraum in Minuten nach dem Eintrag* `thisUpdate` *der OCSP-Antwort, für den eine Antwort als gültig betrachtet wird. Ist die Antwort älter als der angegebene Zeitraum (erweitert um* `maxclockskew`*), wird sie als ungültig betrachtet und nicht verwendet. Standardwert: 1440 (1 Tag)*
hash	*(Schlüsselwort) Hash-Algorithmus zur Identifizierung des Zertifikats in allen OCSP-Anfragen und -Antworten. Der Algorithmus muss von einem OCSP-Responder unterstützt werden (Standardwert:* `sha1`*):* `sha1`*,* `sha256`*,* `sha384` *oder* `sha512` *Da Acrobat XI/DC nur* `sha1` *unterstützen, erfordern aller anderen Werte* `conformance=extended`*.*
maxclockskew	*(Integer) Maximale Toleranz in Minuten bei der Prüfung, ob die Antwort gemäß dem Eintrag* `thisUpdate` *in der OCSP-Antwort und der Option* `freshness` *aktuell genug ist. Diese Option kann verwendet werden, um Netzwerkverzögerungen oder ungenauen Systemzeit zu kompensieren. Standardwert: 5*
nonce	*(Boolean) Bei* `true` *wird die Erweiterung* `nonce` *(»number used only once«) allen OCSP-Anfragen hinzugefügt. Der gleiche Wert muss auch in den OCSP-Antworten vorhanden sein. Die* `Nonce`*-Verarbeitung vereitelt zwar Replay-Angriffe, verhindert aber auch Caching und wird daher von einigen OCSP-Respondern nicht unterstützt. Standardwert:* `true`

Tabelle 8.11 Unteroptionen der Option ocsp *von PLOP_prepare_signature( )*

source	*(Netzwerk-Optionsliste) Optionsliste, die einen Server beschreibt, von dem eine OCSP-Antwort für das Signaturzertifikat angefordert und dann in die Signatur oder den DSS eingebettet wird. Die Protokolle* http *und* https *werden unterstützt. Die Unteroption* url *der Option* source *oder die Option* source *selbst können weggelassen werden. In diesem Fall wird die URL der Erweiterung* authorityInfoAccess *(AIA) in der digitalen ID entnommen.* *Alle Unteroptionen der Netzwerkoptionsliste* source *außer* url *und* httpauthentication *werden auch auf CRL-Anfragen für Zertifikate (außer Signaturzertifikaten) angewendet. Dies erleichtert die Verwendung der gleichen Anmeldeinformationen (z.B.* username/password*) in OCSP-Aufrufen für alle beteiligten Zertifikate.*

Netzwerk-Optionslisten. Verschiedene Funktionen erfordern Zugriff auf ein Netzwerk-Ressourcen wie TSAs und OCSP-Responder. Der Server sowie eventuell weitere Angaben zum Zugriff auf den Server werden in einer Netzwerk-Optionsliste mit den Unteroptionen gemäß Tabelle 8.12 angegeben. Jede Option vom Datentyp »Netzwerk-Optionsliste« gibt die Liste der unterstützten Protokolle an. Beispiele für die Verwendung von Netzwerk-Optionslisten (diese sind blau dargestellt):

```
timestamp={source={url={http://timestamp.acme.com/}} hash=sha384} digitalid=...

ocsp={source={url={http://ocsp.acme.com/}} } digitalid=...

ocsp={source={timeout=1000}} digitalid=...
```

Tabelle 8.12 Unteroptionen für eine Netzwerk-Optionsliste

Option	*Beschreibung*
httpauthentication	*(Schlüsselwort; nur für* `http`*) Mögliche Authentisierungsmethode(n). Ein Server unterstützt eventuell nur bestimmte (oder gar keine) Authentisierungsmethoden. Zur Verbesserung der Leistung sollten Sie statt des Standardwerts die Authentisierungsmethode explizit angeben (auch wenn dadurch die gleiche Methode ausgewählt wird). Unterstützte Schlüsselwörter (Standardwert:* `any`*):* ***any*** *Die sicherste vom Server unterstützte Authentisierungsmethode wird verwendet.* ***anysafe*** *Wie* `any`*, jedoch ohne Basic Authentication.* ***basic*** *Basic Authentication mit Benutzername und Kennwort. Diese Methode ist nicht empfehlenswert, da Benutzername und Kennwort dabei im Klartext über das Netz geschickt werden.* ***digest*** *Digest Authentication mittels Hash vom Benutzernamen und Kennwort gemäß RFC 2617. Diese Methode ist sicherer als Basic Authentication.* ***ntlm*** *NTLM Authentication, wie in den Produkten von Microsoft verwendet*
password	*(String) Kennwort für Basic und Digest Authentication*
sslcertdir	*(String; nur für* `https`*) Name eines Verzeichnisses, das vertrauenswürdige CA-Zertifikate in PEM-Kodierung enthält, die zum Aufbau einer SSL-Verbindung erforderlich sind. Für Dateinamenskonventionen siehe »Namenskonventionen für Zertifikate und CRL-Dateien«, Seite 179.*
sslcertfile	*(String; nur für* `https`*) Name eines Verzeichnisses, das vertrauenswürdige CA-Zertifikate in PEM-Kodierung enthält, die zum Aufbau einer SSL-Verbindung erforderlich sind.*
sslverifyhost	*(Boolean; nur für* `https`*) Bei* `true` *muss das Feld* `Subject Alternate Name` *im Serverzertifikat mit dem Host-Namen in der URL übereinstimmen, um eine Verbindung herzustellen. Standardwert:* `true`
sslverifypeer	*(Boolean; nur für* `https`*) Bei* `true` *muss das Serverzertifikat gegen die vertrauenswürdigen Zertifikate verifizierbar sein, die mit den Optionen* `sslcertdir` *oder* `sslcertfile` *übergeben werden. Bei* `false` *wird auch ein Serverzertifikat akzeptiert, das nicht geprüft werden kann, weil kein vertrauenswürdiges Stammzertifikat dafür vorliegt. Standardwert:* `true`
timeout	*(Integer) Zeitlimit für den Zugriff auf die Ressource in Millisekunden. Bei 0 wird kein Zeitlimit angewendet. Standardwert: 15000*
url	*(String; meist erforderlich, aber optional, wenn die URL aus dem Kontext bekannt ist) Vollständig qualifizierte URL der Netzwerk-Ressource einschließlich der führenden Protokollidentifikation. Die unterstützten Protokolle sind in der Beschreibung der jeweiligen Option angegeben, die eine Netzwerk-Optionsliste verwendet. Zeichen können in URL-Kodierung angegeben werden, z.B. %20. Die URL kann den Benutzernamen und das Kennwort in Standardsyntax enthalten, z.B.* `http://user:password@timestamp.acme.com/`
username	*(String) Benutzername für Basic und Digest Authentication*

8.7 Verarbeitung von Exceptions

PLOP bietet Hilfsmethoden zur Verarbeitung von Bibliothek-Exceptions in der Programmiersprache C. Andere PLOP-Sprachbindungen verwenden das in der jeweiligen Sprachbindung integrierte System zur Verarbeitung von Exceptions, wie zum Beispiel die *try/catch*-Klauseln. Die Sprach-Wrapper fügen Informationen über Exception-Nummer, Beschreibung und API-Funktionsnamen in das generierte Exception-Objekt ein.

Bei einer PLOP-Exception kann keine andere PLOP-Funktion mit dem zugehörigen PLOP-Objekt aufgerufen werden außer *PLOP_delete( )*, *PLOP_get_errnum( )*, *PLOP_get_errmsg( )*, *PLOP_get_apiname( )* .

Die PLOP-Sprachbindungen für Java und .NET verwenden ein separates Objekt *PLOPException*, dessen Member detaillierte Informationen zur Fehlerursache enthalten.

C++ Java ***int get_errnum( )***

Perl PHP ***int get_errnum( )***

C ***int PLOP_get_errnum(PLOP *plop)***

Ermittelt die Nummer der zuletzt ausgelösten Exception oder die Ursache für einen gescheiterten Funktionsaufruf.

Rückgabe Die Fehlernummer der Exception.

Bindungen In .NET ist diese Methode auch als *Errnum* im Objekt *PLOPException* verfügbar.
In Java ist diese Methode auch als *get_errnum( )* im Objekt *PLOPException* verfügbar.

C++ Java ***String get_errmsg( )***

Perl PHP ***string get_errmsg( )***

C ***const char *PLOP_get_errmsg(PLOP *plop)***

Ermittelt die Beschreibung der zuletzt ausgelösten Exception oder die Ursache für einen gescheiterten Funktionsaufruf.

Rückgabe String mit der Fehlerbeschreibung oder ein leerer String, wenn der letzte API-Aufruf keinen Fehler ausgelöst hat.

Bindungen In .NET ist diese Methode auch als *Errmsg* im Objekt *PLOPException* verfügbar.
In Java ist diese Methode auch als *getMessage( )* im Objekt *PLOPException* verfügbar.

C++ Java ***String get_apiname( )***

Perl PHP ***string get_apiname( )***

C ***const char *PLOP_get_apiname(PLOP *plop)***

Ermittelt den Namen der API-Funktion, die die letzte Exception ausgelöst hat oder scheiterte.

Rückgabe Name einer PLOP-API-Funktion.

Bindungen In .NET ist diese Methode auch als *Apiname* im Objekt *PLOPException* verfügbar.
In Java ist diese Methode auch als *get_apiname( )* im Objekt *PLOPException* verfügbar.

C ***PLOP_TRY(PLOP *plop)***

Richtet einen Rahmen für die Verarbeitung von Exceptions ein; muss immer paarweise mit *PLOP_CATCH( )* aufgerufen werden.

Details Siehe »Fehlerbehandlung«, Seite 45.

C ***PLOP_CATCH(PLOP *plop)***

Fängt eine Exception ab; muss immer paarweise mit *PLOP_TRY( )* aufgerufen werden.

Details Siehe »Fehlerbehandlung«, Seite 45.

C ***PLOP_EXIT_TRY(PLOP *plop)***

Informiert die Exception-Verarbeitung, dass ein Block mit *PLOP_TRY( )* ohne den Aufruf von *PLOP_CATCH( )* verlassen wird.

Details Siehe »Fehlerbehandlung«, Seite 45.

C ***PLOP_RETHROW(PLOP *plop)***

Reicht die Exception an ein anderen Handler weiter.

Details Siehe »Fehlerbehandlung«, Seite 45.

8.8 Globale Optionen

C++ Java ***void set_option(String optlist)***

Perl PHP ***set_option(string optlist)***

C ***void PLOP_set_option(PLOP *plop, const char *optlist)***

Setzt eine oder mehrere globale Optionen für PLOP.

optlist Optionsliste mit globalen Optionen gemäß Tabelle 8.13. Wird eine Option mehrfach übergeben, überschreibt der letzte Wert alle früheren. Um einer Option mehrere Werte mitzugeben (z.B. *searchpath),* übergeben Sie alle Werte in einem Listenargument.

Details Mit mehrfachen Aufrufen dieser Funktion können Werte für die in Tabelle 8.13 markierten Optionen akkumuliert werden. Bei nicht markierten Optionen überschreibt der neue Wert den alten.

Tabelle 8.13 Globale Optionen für PLOP_set_option( )

Option	Beschreibung
filename-handling	*(Schlüsselwort) Encoding von Dateinamen. Als Funktionsparameter ohne UTF-8-BOM in nicht Unicode-fähigen Sprachbindungen übergebene Dateinamen werden gemäß dieser Option interpretiert, um im Dateisystem nicht zulässige Zeichen zu vermeiden und um eine Unicode-Version der Dateinamen zu erzeugen. Enthält der Dateiname Zeichen außerhalb des angegebenen Encodings, wird eine Exception ausgelöst. Standardwert:* `unicode` *für Windows und OS X/macOS, sonst* `honorlang`*:* ***ascii*** *7-Bit-ASCII* ***basicebcdic*** *EBCDIC gemäß Codepage 1047, aber nur mit Unicode-Werten <= U+007E* ***basicebcdic_37*** *EBCDIC gemäß Code-Page 0037, aber nur Unicode-Werte <= U+007E* ***honorlang*** *Die Umgebungsvariablen LC_ALL, LC_CTYPE und LANG werden ausgewertet. Sofern vorhanden, wird der in* `LANG` *angegebene Codeset auf die Dateinamen angewendet.* ***legacy*** *Verwendet das Encoding* `auto` *(d.h. das aktuelle System-Encoding) zur Interpretation des Dateinamens und interpretiert die Variable LANG, sofern der Parameter* `honorlang` *gesetzt ist.* ***unicode*** *Unicode-Encoding im (EBCDIC-)UTF-8-Format* ***all Namen von 8-Bit- und CJK-Encodings*** *Beliebiges von PLOP erkanntes Encoding*
license	*(String) Setzt den Lizenzschlüssel. Diese Option muss vor dem ersten Aufruf von PLOP_open_document*( ) gesetzt werden.*
licensefile	*(String) Setzt den Namen einer Datei mit Lizenzschlüsseln. Die Lizenzdatei kann nur einmal vor dem ersten Aufruf von PLOP_open_document*( ) festgelegt werden. Alternativ kann der Name der Lizenzdatei in einer Umgebungsvariablen namens PDFLIBLICENSEFILE oder unter Windows in der Registry übergeben werden.*
frontpage	*(Boolean) Bei* `false` *wird eine Exception ausgelöst, wenn kein gültiger Lizenzschlüssel gefunden wurde; bei* `true` *wird im Evaluierungsmodus eine Titelseite gemäß Abschnitt 0.1, »Installation der Software«, Seite 7 erzeugt. Diese Option muss vor dem ersten Aufruf von PLOP_open_document*( ) gesetzt werden. Sie hat keine Auswirkung, wenn ein gültiger Lizenzschlüssel gefunden wurde. Standardwert:* `true`

Tabelle 8.13 Globale Optionen für PLOP_set_option( )

Option	Beschreibung
logging[1]	*(Optionsliste; nicht unterstützt) Optionsliste zur Steuerung der Logging-Ausgabe gemäß Tabelle 8.15. Alternativ können Logging-Optionen in einer Umgebungsvariablen namens PLOPLOGGING oder unter Windows in der Registry übergeben werden. Mit einer leeren Optionsliste kann die Protokollierung mit den in vorigen Aufrufen gesetzten Optionen aktiviert werden. Ist die Umgebungsvariable gesetzt, startet die Protokollierung direkt nach dem ersten Aufruf von PLOP_new( ).*
mmiolimit	*(Integer) Oberes Limit für die Größe der Eingabedateien in MB (=1024*1024 Bytes), die im Speicher gemapped sind. Wird diese Option auf 0 (null) gesetzt, wird das Speicher-Mapping deaktiviert. Deaktiviertes Speicher-Mapping kann auf Nicht-Windows-Systemen verwendet werden, um Probleme zu vermeiden, wenn entfernt gespeicherte Dateien während der Verwendung plötzlich nicht mehr verfügbar sind. Standardwert: 50 auf 32-Bit-Plattformen, sonst 2048*
searchpath[1]	*(Liste von Name-Strings) Relative oder absolute Pfadname(n) eines Verzeichnisses mit zu lesenden Dateien. Searchpath kann mehrfach gesetzt werden; die Einträge werden in einer Liste verwaltet, die von hinten nach vorne abgearbeitet wird. Es wird empfohlen, selbst für einen einzigen Eintrag doppelt geschweifte Klammern zu verwenden, um Probleme mit Leerzeichen in Verzeichnisnamen zu vermeiden. Eine leere String-Liste (d.h. {{}}) löscht alle bestehenden Searchpath-Einträge einschließlich der Standardeinträge. Bei Windows kann der Searchpath auch über die Registry gesetzt werden. Standardwert: leer*
userlog	*(Name-String) Beliebiger String, der in die Logdatei geschrieben wird, wenn die Protokollierung aktiviert ist.*

1. Optionswerte können mit Mehrfach-Aufrufen akkumuliert werden.

8.9 Logging (Protokollierung)

Mit der Logging-Funktion lassen sich API-Aufrufe protokollieren. Der Inhalt der Logdatei kann bei der Fehlersuche hilfreich sein oder vom Support der PDFlib GmbH benötigt werden. Tabelle 8.15 führt Optionen zur Aktivierung von Logging-Funktionen mit *PLOP_set_option( )* auf.

Tabelle 8.14 Logging-spezifische Optionen für PLOP_set_option( )

Option	Beschreibung
logging	*Optionsliste mit Logging-Optionen gemäß Tabelle 8.15*
userlog	*String, der in die Logdatei geschrieben wird*

- Die Logging-Optionen können auf folgende Arten übergeben werden:
- Als Optionsliste für die Option *logging* von *PLOP_set_option( )*, z.B.:

```
plop.set_option("logging={filename={debug.log} remove}");
```

- In der Umgebungsvariablen *PLOPLOGGING*. Dies aktiviert die Protokollierung ab dem ersten Aufruf einer API-Funktion.

Tabelle 8.15 Unteroptionen für die `logging`*-Option von PLOP_set_option( )*

Option	Beschreibung
classes	*(Optionsliste) Liste mit Optionen, wobei jede Option eine Logging-Klasse und der zugehörige Wert den Logging-Level beschreibt. Level 0 deaktiviert eine Logging-Klasse; positive Zahlen aktivieren eine Klasse. Je höher der Level ist, desto detaillierter ist die Ausgabe. Ist kein Level für eine Klasse angegeben, wird der Wert 1 verwendet: (erster Wert:* `api=1`*).* **api** *Protokolliert alle API-Funktionsaufrufe mit Parametern und Rückgabewerten. Bei* `api=2` *wird ein Zeitstempel vor jeden API-Funktionsaufruf gestellt, und veraltete Funktionen und Optionen werden markiert.* **digsig** *Protokolliert Details zur Erstellung von digitalen Signaturen:* **1** *Allgemeine Informationen* **2** *Gültigkeitsinformationen; Details zu OCSP und CRL; Informationen zu PKCS#11-Bibliothek, Slot und Token* **5** *Details zu Zertifikaten* **filesearch** *Protokolliert alle Versuche, Dateien via* `SearchPath` *oder PVF zu finden.* **resource** *Protokolliert alle Versuche, Ressourcen über die Windows-Registry, UPR-Definitionen oder Ergebnisse der Ressourcensuche zu finden.* **user** *Benutzerdefinierte Logging-Ausgabe, die mit der Option* `userlog` *übergeben wurde.* **warning** *Protokolliert alle Warnungen, d.h. Fehlerbedingungen, die ignoriert oder intern behandelt werden können. Ist* `warning=2`*, werden auch Meldungen von Funktionen protokolliert, die zwar keine Exception auslösen, aber den Meldungstext zur Abfrage mit PLOP_get_errmsg( ) liefern, sowie die Ursache für alle gescheiterten Versuche, eine Datei zu öffnen (Suche einer Datei via* `searchpath`*).*
disable	*(Boolean) Protokollierung deaktivieren. Standardwert:* `false`

Tabelle 8.15 Unteroptionen für die `logging`*-Option von PLOP_set_option( )*

Option	*Beschreibung*
filename	*(String) Name der Logdatei (*`stdout` *und* `stderr` *werden auch akzeptiert). Die Ausgabe wird an bereits vorhandene Inhalte angehängt. Der Name der Logdatei kann auch in einer Umgebungsvariable namens PLOPLOGFILENAME übergeben werden (in diesem Fall wird die Option* `filename` *immer ignoriert). Standardwert:* `plop.log` *(unter Windows und OS X/macOS im Verzeichnis /, unter Unix in /tmp)*
flush	*(Boolean) Bei* `true` *wird die Logdatei nach jeder Ausgabe geschlossen und bei der nächsten Ausgabe erneut geöffnet. Damit ist gewährleistet, dass die Ausgabe sicher in die Datei geschrieben wird. Dies kann bei der Verfolgung von Programmabstürzen mit unvollständiger Logdatei nützlich sein. Die Verarbeitungsgeschwindigkeit verringert sich jedoch erheblich. Bei* `false` *wird die Logdatei nur einmal geöffnet. Standardwert:* `false`
includepid	*(Boolean; nicht für MVS) Angabe der Prozess-ID im Logdateinamen. Dies sollte aktiviert werden, wenn mehrere Prozesse den selben Logdateinamen verwenden. Standardwert:* `false`
includetid	*(Boolean; nicht für MVS) Angabe der Thread-ID im Logdateinamen. Dies sollte aktiviert werden, wenn mehrere Threads im selben Prozess den selben Logdateinamen verwenden. Standardwert:* `false`
includeoid	*(Boolean; nicht für MVS) Angabe der Objekt-ID im Logdateinamen. Dies sollte aktiviert werden, wenn mehrere PLOP-Objekte im selben Thread den selben Logdateinamen verwenden. Standardwert:* `false`
remove	*(Boolean) Bei* `true` *wird eine gegebenenfalls bereits vorhandene Logdatei gelöscht, bevor die neue Ausgabe geschrieben wurde. Standardwert:* `false`
removeonsuccess	*(Boolean) Entfernt die generierte Logdatei in PLOP_delete( ), sofern keine Exception ausgelöst wird. Dies kann für die Analyse gelegentlicher Probleme in Multi-Threaded-Anwendungen oder bei sporadisch auftretenden Problemen nützlich sein. Wir empfehlen, diese Option nach Bedarf mit* `includepid`*/*`includetid`*/*`includeoid` *zu kombinieren.*
restore	*(Boolean) Stellt den Status aller Ebenen von Logging-Klassen (außer denen in der selben Optionsliste) bis zum letzten gespeicherten Stand wieder her.*
save	*(Boolean) Speichert den Status aller Ebenen von Logging-Klassen (außer denen in der selben Optionsliste). Bis zu 7 Speicherebenen werden unterstützt.*
stringlimit	*(Integer) Limit für die Anzahl der Zeichen in Text-Strings oder 0 für kein Limit. Standardwert: 0*

8.10 pCOS-Funktionen

Die pCOS-Syntax zur Abfrage von Objektdaten aus einem PDF wird vollständig unterstützt; für weitere Informationen siehe die pCOS-Pfadreferenz.

C++ Java ***double pcos_get_number(int doc, String path)***

Perl PHP ***double pcos_get_number(long doc, string path)***

C ***double PLOP_pcos_get_number(PLOP *plop, int doc, const char *path, ...)***

Liefert den Wert eines pCOS-Pfades vom Typ *Zahl* oder *Boolean*.

doc Gültiges Dokument-Handle, das mit *PLOP_open_document*()* erzeugt wurde.

path Vollständiger pCOS-Pfad für ein Zahl- oder Boolean-Objekt.

Weitere Parameter (Nur C-Sprachbindung) Es können ein oder mehrere zusätzliche Parameter übergeben werden, sofern der Parameter *key* entsprechende Platzhalter enthält *(%s* für Strings oder *%d* für Integers; *%%* wird für ein einzelnes Prozentzeichen verwendet). Diese Parameter ersparen es Ihnen, komplexe Pfade mit mehreren Zahlen oder Strings explizit zu formatieren. Der Client muss sicherstellen, dass Anzahl und Typ der Platzhalter mit den zusätzlich übergebenen Parametern übereinstimmen.

Rückgabe Numerischer Wert des durch den pCOS-Pfad bezeichneten Objekts. Bei Booleschen Werten wird 1 zurückgegeben, wenn sie *true* sind, andernfalls 0.

C++ Java ***string pcos_get_string(int doc, String path)***

Perl PHP ***string pcos_get_string(long doc, string path)***

C ***const char *PLOP_pcos_get_string(PLOP *plop, int doc, const char *path, ...)***

Liefert den Wert eines pCOS-Pfades vom Typ *Name, Zahl, String* oder *Boolean*.

doc Gültiges Dokument-Handle, das mit *PLOP_open_document*()* erzeugt wurde.

path Vollständiger pCOS-Pfad für ein String-, Zahl-, Name- oder Boolean-Objekt.

Weitere Parameter (Nur C-Sprachbindung) Es können ein oder mehrere zusätzliche Parameter übergeben werden, sofern der Parameter *key* entsprechende Platzhalter enthält *(%s* für Strings oder *%d* für Integers; *%%* wird für ein einzelnes Prozentzeichen verwendet). Diese Parameter ersparen es Ihnen, komplexe Pfade mit mehreren Zahlen oder Strings explizit zu formatieren. Der Client muss sicherstellen, dass Anzahl und Typ der Platzhalter mit den zusätzlich übergebenen Parametern übereinstimmen.

Rückgabe String mit dem Wert des durch den pCOS-Pfad bezeichneten Objekts. Bei Booleschen Werten wird einer der Strings *true* oder *false* zurückgegeben.

Details Diese Funktion löst eine Exception aus, wenn pCOS nicht im vollständigen Modus läuft und der Typ des Objekts *string* ist. Die Objekte */Info/** (Dokument-Infoschlüssel) können jedoch auch im eingeschränkten pCOS-Modus abgefragt werden, sofern *nocopy=false* oder *plainmetadata=true* ist; und *bookmarks[...]/Title* sowie alle Pfade beginnend mit *pages[...]/annots[...]/* können im eingeschränkten pCOS-Modus abgefragt werden, sofern *nocopy=false* ist.

Diese Funktion geht davon aus, dass die Strings, die aus dem PDF-Dokument abgefragt werden, Text-Strings sind. String-Objekte, die Binärdaten enthalten, sollten mit *PLOP_pcos_get_stream( )* abgefragt werden, das die Daten unverändert zurückliefert.

Bindungen C-Sprachbindung: Der String wird im UTF-8-Format ohne BOM zurückgegeben. Die zurückgegebenen Strings werden in einem Ring-Puffer mit bis zu 10 Einträgen gespeichert. Werden mehr als 10 Strings abgefragt, werden die Puffer wiederverwendet. Clients müssen die Strings deshalb kopieren, wenn sie auf mehr als 10 Strings gleichzeitig zugreifen möchten. Bis zu 10 Aufrufe dieser Funktion können zum Beispiel als Parameter für eine *printf( )*-Anweisung verwendet werden, da die Rückgabe-Strings unabhängig voneinander sind, sofern nicht mehr als 10 Strings gleichzeitig verwendet werden.

C++-Sprachbindung: Der String wird in der standardmäßig aktiven *wstring*-Konfiguration des C++-Wrappers als *wstring* zurückgegeben. Im Kompatibilitätsmodus *string* unter zSeries wird das Ergebnis im EBCDIC-UTF-8-Format ohne BOM zurückgegeben.

Java- und .NET-Sprachbindungen: das Ergebnis wird als Unicode-String zurückgegeben.

Perl-, PHP-, Python- und Ruby-Sprachbindungen: das Ergebnis wird als UTF-8-String zurückgegeben.

C++ ***const unsigned char *pcos_get_stream(int doc, int *length, string optlist, wstring path)***

C# Java ***byte[ ] pcos_get_stream(int doc, String optlist, String path)***

Perl PHP ***string pcos_get_stream(long doc, string optlist, string path)***

C ***const unsigned char *PLOP_pcos_get_stream(PLOP *plop, int doc, int *length, const char *optlist, const char *path, ...)***

Liefert den Inhalt eines pCOS-Pfades vom Typ *stream, fstream* oder *string*.

doc Gültiges Dokument-Handle, das mit *PLOP_open_document*( )* erzeugt wurde.

length (Nur C- und C++-Sprachbindung) Zeiger auf eine Variable zur Speicherung der Länge der zurückgegebenen Streamdaten in Bytes.

optlist Optionsliste mit Optionen zur Abfrage von Streamdaten gemäß Tabelle 8.16.

path Vollständiger pCOS-Pfad für ein Stream- oder String-Objekt.

Weitere Parameter (Nur C-Sprachbindung) Es können ein oder mehrere zusätzliche Parameter übergeben werden, sofern der Parameter *key* entsprechende Platzhalter enthält *(%s* für Strings oder *%d* für Integers; *%%* wird für ein einzelnes Prozentzeichen verwendet). Diese Parameter ersparen es Ihnen, komplexe Pfade mit mehreren Zahlen oder Strings explizit zu formatieren. Der Client muss sicherstellen, dass Anzahl und Typ der Platzhalter mit den zusätzlich übergebenen Parametern übereinstimmen.

Rückgabe Die im Stream bzw. String enthaltenen unverschlüsselten Daten. Die zurückgegebenen Daten sind leer (in C und C++: NULL), wenn der Stream bzw. String leer ist oder wenn Inhalte verschlüsselter Anhänge in einem unverschlüsselten Dokument abgefragt werden und für die angehängten Dokumente kein Kennwort übergeben wurde.

Ist das Objekt vom Typ *stream*, werden alle Filter vom Stream entfernt (d.h. die eigentlichen Rohdaten werden zurückgegeben). Ist das Objekt vom Typ *fstream* oder

string, werden die Daten so zurückgegeben, wie sie im PDF vorgefunden wurden, mit Ausnahme der Filter ASCII85 und ASCIIHex, die entfernt werden.

Details Diese Funktion löst eine Exception aus, wenn pCOS nicht im vollständigen Modus läuft. Eine Ausnahme bildet das Objekt */Root/Metadata*, das auch im eingeschränkten pCOS-Modus abrufbar ist, sofern *nocopy=false* oder *plainmetadata=true*. Eine Exception wird zudem ausgelöst, wenn *path* nicht auf ein Objekt vom Typ *stream*, *fstream* oder *string* zeigt.

Ungeachtet ihres Namens kann diese Funktion auch zur Abfrage von Objekten vom Typ *string* eingesetzt werden. Im Gegensatz zu *PLOP_pcos_get_string( )*, das das Objekt als Text-String behandelt, verändert diese Funktion die zurückgegebenen Daten in keiner Weise. Strings mit Binärdaten werden in PDF selten verwendet und lassen sich nicht zuverlässig automatisch erkennen. Benutzer müssen deshalb selbst darauf achten, bei der Abfrage von String-Objekten die für Binärdaten oder Text geeignete Funktion zu verwenden.

Bindungen COM: Client-Programme verwenden zur Speicherung des Stream-Inhalts meist den Typ *Variant*. Bei JavaScript mit COM ist es nicht möglich, die Länge des zurückgegebenen Varianten-Arrays abzufragen (bei anderen Sprachen mit COM ist dies aber möglich).

C- und C++-Sprachbindungen: Der zurückgegebene Datenpuffer kann bis zum nächsten Aufruf dieser Funktion verwendet werden.

Mit dieser Funktion lassen sich eingebettete Fonts aus PDF extrahieren. Beachten Sie, dass Fonts den Lizenzvereinbarungen der jeweiligen Hersteller unterliegen und ohne explizite Genehmigung des Rechteinhabers nicht weiterverwendet werden dürfen. Kontaktieren Sie gegebenenfalls den Anbieter des Fonts, um Lizenzfragen zu klären.

Tabelle 8.16 Option für PLOP_pcos_get_stream( )

Option	Beschreibung	
convert	*(Schlüsselwort; wird bei Streams ignoriert, die mit nicht unterstützten Filtern komprimiert sind) Steuert, ob die String- oder Stream-Inhalte konvertiert werden (Standardwert:* none*)*:	
	none	*Inhalte werden als Binärdaten ohne Konvertierung behandelt.*
	unicode	*Inhalte werden als Textdaten behandelt (d.h. genauso wie in PLOP_pcos_get_string( )) und nach Unicode normalisiert. In nicht Unicode-fähigen Sprachbindungen werden Daten ins Format UTF-8 ohne BOM formatiert.* *Diese Option wird für den selten verwendeten PDF-Datentyp »Text Stream« benötigt (er kann z.B. für JavaScript verwendet werden, obwohl der Großteil der JavaScripts in String- und nicht in Stream-Objekten enthalten ist).*

8.11 Funktion zur Unicode-Konvertierung

C++ ***string convert_to_unicode(wstring inputformat, string input, wstring optlist)***

C# Java ***string convert_to_unicode(String inputformat, byte[ ] input, String optlist)***

Perl PHP ***string convert_to_unicode(string inputformat, string input, string optlist)***

C ***const char *PLOP_convert_to_unicode(PLOP *p,***
const char *inputformat, const char *input, int inputlen, int *outputlen, const char *optlist)

Konvertiert einen String mit beliebigem Encoding in einen Unicode-String mit wählbarem Format.

inputformat Unicode-Textformat oder Name des Encodings des zu konvertierenden Strings:

- Unicode-Textformate: *utf8, ebcdicutf8, utf16, utf16le, utf16be, utf32*
- Alle intern bekannten 8-Bit-Encodings, auf dem Host-System vorhandene Encodings sowie die CJK-Encodings *cp932, cp936, cp949, cp950*
- Das Schlüsselwort *auto* führt zu folgendem Verhalten: wenn der Eingabe-String einen UTF-8-BOM oder UTF-16-BOM enthält, wird dieser zur Bestimmung des korrekten Formats verwendet, ansonsten wird die aktuelle Codepage des Systems herangezogen.

input (String: bei COM: Variant) Nach Unicode zu konvertierender String.

inputlen (Nur C-Sprachbindung) Länge des zu konvertierenden Strings in Bytes. Ist *inputlen=0,* muss ein null-terminierter String übergeben werden.

outputlen (Nur C-Sprachbindung) C-Zeiger auf einen Speicherplatz, an dem die Länge des zurückgegebenen Strings in Bytes abgelegt wird.

optlist Optionsliste, in der die Optionen für die Interpretation des zu konvertierenden Strings und für seine Konvertierung nach Unicode festgelegt werden:

- Optionen zu Eingabefiltern gemäß Tabelle 8.17: *charref, escapesequence*
- Optionen für Unicode-Konvertierung gemäß Tabelle 8.17: *bom, errorpolicy, inflate, outputformat*

Rückgabe Ein aus dem zu konvertierenden String gemäß den angegebenen Parametern und Optionen erzeugter Unicode-String. Wenn der zu konvertierende String nicht dem angegebenen Eingabeformat entspricht (z.B. bei einem ungültigen UTF-8-String), wird bei *errorpolicy=return* ein leerer String zurückgegeben und bei *errorpolicy=exception* wird eine Exception ausgelöst.

Details Diese Funktion ist für die Unicode-Konvertierung von Strings nützlich, besonders wenn Sie in einer Umgebung ohne geeignete Unicode-Konverter arbeiten.

Bindungen C-Sprachbindung: Bis zu 10 zurückgegebene String-Einträge werden in einem Ring-Puffer abgelegt. Werden mehr als 10 Strings konvertiert, werden die Puffer wiederverwendet. Clients müssen die Strings deshalb kopieren, wenn sie auf mehr als 10 Strings gleichzeitig zugreifen möchten. Bis zu 10 Aufrufe dieser Funktion können zum Beispiel als Parameter für eine *printf( )*-Anweisung verwendet werden, da die Rückgabe-Strings unabhängig voneinander sind, sofern nicht mehr als 10 Strings gleichzeitig verwendet werden.

Tabelle 8.17 Optionen für PLOP_convert_to_unicode( )

Option	Beschreibung
bom	*(Schlüsselwort; wird bei* `outputformat=utf32` *nicht ausgewertet) Richtlinie zum Hinzufügen eines Byte Order Mark (BOM) zum Ausgabe-String. Unterstützte Schlüsselwörter (Standardwert:* `none`*):* ***add*** *Hinzufügen eines BOM.* ***keep*** *Hinzufügen eines BOM, wenn der Eingabe-String einen BOM hat.* ***none*** *Kein Hinzufügen eines BOM.* ***optimize*** *Hinzufügen eines BOM, außer wenn* `outputformat=utf8` *oder* `ebcdicutf8` *und der Ausgabe-String nur Zeichen kleiner U+007F enthält.*
charref	*(Boolean) Bei* `true` *werden numerische Referenzen, Character-Referenzen und Glyphnamen-Referenzen ersetzt. Standardwert:* `false`
errorpolicy	*(Schlüsselwort) Verhalten bei einem Konvertierungsfehler (Standardwert:* `exception`*):* ***return*** *Das Ersatzzeichen wird verwendet, wenn eine Character-Referenz nicht aufgelöst werden kann. Bei einem Konvertierungsfehler wird ein leerer String ausgegeben.* ***exception*** *Bei einem Konvertierungsfehler wird eine Exception ausgelöst.*
escape-sequence	*(Boolean) Bei* `true` *werden Escape-Sequenzen in Strings ersetzt. Standardwert:* `false`
inflate	*(Boolean; nur bei* `inputformat=utf8`*; wird bei* `outputformat=utf8` *nicht ausgewertet) Bei* `true` *löst ein ungültiger UTF-8-Eingabe-String keine Exception aus, vielmehr wird ein String erzeugt, der die Bytes des Eingabe-Strings als Unicode-Zeichen enthält. Dies kann bei der Fehlersuche hilfreich sein. Standardwert:* `false`
outputformat	*(Schlüsselwort) Unicode-Textformat des generierten Strings:* `utf8, ebcdicutf8, utf16, utf16le, utf16be, utf32.` *Ein leerer String ist äquivalent zu* `utf16`*. Standardwert:* `utf16` *Unicode-fähige Sprachbindungen: das Ausgabeformat wird immer auf* `utf16` *gesetzt.* *C++-Sprachbindung: nur die folgenden Ausgabeformate sind erlaubt:* `utf8, utf16, utf32.`

A Arbeiten mit Zertifikaten

In diesem Anhang finden Sie zusätzliche nützliche Informationen für den Einsatz von Zertifikaten mit PLOP oder PLOP DS.

Anzeigen von Zertifikatinhalten. Inhalte eines Zertifikats oder einer digitalen ID lassen sich mit folgendem Windows-Befehl anzeigen:

```
certutil -dump -p demo demo_signer_rsa_2048.p12
```

Anzeige von Zertifikatinhalten mit Open SSL:

```
openssl x509 -inform DER -in demo_recipient_1.pem -noout -text
```

Extrahieren des öffentlichen Schlüssels aus einer digitalen ID zur Erzeugung eines Zertifikats mit OpenSSL. Wenn Sie eine digitale ID haben (mit öffentlichem und privatem Schlüssel) und für andere ein passendes Zertifikat (mit privatem Schlüssel) benötigen, um Dokumente zu verschlüsseln, können Sie folgenden Befehl verwenden:

```
openssl pkcs12 -in demo_signer_rsa_2048.p12 -clcerts -nokeys -out ↵
        demo_signer_rsa_2048.pem
```

Konvertieren eines Zertifikats nach PEM mit OpenSSL. Die Signaturoptionen *certfile* und *rootcertfile* sowie die Unteroption *sslcertfile* in einer Netzwerkoptionsliste akzeptieren Zertifikate nur in der textbasierten PEM-Kodierung. Auf EBCDIC-Plattformen müssen PEM-Zertifikate im Format EBCDIC kodiert sein.

Mit folgendem OpenSSL-Befehl lassen sich Zertifikate von der binären DER-Kodierung in die erforderliche textbasierte PEM-Kodierung konvertieren:

```
openssl x509 -inform DER -outform PEM -in PDFlibDemoCA_G2.crt -out PDFlibDemoCA_G2.pem
```

Beachten Sie, dass *.cer*- und *.crt*-Dateien DER- oder PEM-Kodierung verwenden können. Da die Dateinamenserweiterung unzuverlässig ist, können Sie das Format in einem Texteditor überprüfen: Während DER-Kodierung aus binären Daten besteht, ist PEM-Kodierung ein textbasiertes Format mit Base-64-kodierten, von Linien eingeschlossenen Daten.

```
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

Namenskonventionen für Zertifikate und CRL-Dateien. Die Signaturoptionen *crldir* und *rootcertdir* sowie die Unteroption *sslcertdir* in einer Netzwerkoptionsliste akzeptieren den Namen eines Verzeichnisses, in dem nach Zertifikaten oder CRLs gesucht wird. Diese Dateien müssen in der textbasierten PEM-Kodierung gespeichert und gemäß den Hash-Konventionen von OpenSSL 1.0.0 (oder höher) für Dateinamen benannt werden (frühere Versionen verwenden andere Namenskonventionen).

Mit dem OpenSSL-Befehl *c_rehash* lassen sich symbolische Links mit den erforderlichen Hash-Dateinamen für ein oder mehrere Verzeichnisse mit Zertifikaten oder CRLs in PEM-Kodierung erzeugen:

```
c_rehash .
```

Falls Sie Hash-Dateinamen manuell erzeugen wollen, gehen Sie folgendermaßen vor:

- Verwenden Sie das DER-kodierte *Subject*-Feld eines Zertifikats bzw. das DER-kodierte *Issuer*-Feld einer CRL und bestimmen Sie den Hash-Wert mittels SHA-1. Berücksichtigen Sie die ersten vier Bytes des erzeugten Hash-Werts und verwenden Sie die ersten 8 hexadezimalen Ziffern als Basis für den Dateinamen.
  Sie können die Dateinamen-Hashes für einzelne Zertifikate oder CRL-Dateien mit OpenSSL-Befehlen ähnlich den folgenden erzeugen:

```
# Erzeugen des Hash-Dateinamens für ein Zertifikat in PEM-Kodierung
openssl x509 -hash -noout -in PDFlibDemoCA_G2.pem

# Erzeugen des Hash-Dateinamens für ein Zertifikat in DER-Kodierung
openssl x509 -hash -noout -inform DER -in PDFlibDemoCA_G2.crt

# Erzeugen des Hash-Dateinamens für eine CRL in PEM-Kodierung
openssl crl -hash -noout -in PDFlibDemoCA_G2.crl.pem

# Erzeugen des Hash-Dateinamens für eine CRL in DER-Kodierung
openssl x509 -hash -noout -inform DER -in PDFlibDemoCA_G2.crt
```

- Hängen Sie folgendes Zeichen an: '.' (Punkt). Hängen Sie für CRLs zusätzlich das Zeichen 'r' an;
- Hängen Sie die Dezimalzahl 0 (null) an. Bei einem Konflikt mit einem bestehenden Hash-Wert müssen Sie statt der Null eine höhere Dezimalzahl anhängen.

Syntax für Object Identifiers (OIDs). Die Unteroption *oid* der Option *policy* und die Unteroption *policy* der Option *timestamp* erwarten einen Object Identifier (OID), der die Richtlinie *(policy)* angibt. OIDs bestehen aus einer Reihe von Dezimalzahlen, die durch Leerzeichen oder einen Punkt ».« voneinander getrennt sind, z.B.

```
2.16.840.1.101.3.2.1.48.9
```

B Kombination von PDFlib mit PLOP DS

PLOP DS wurde für den einfachen Einsatz mit PDFlib zur dynamischen Erzeugung und zum Signieren von PDF-Dokumenten entwickelt. In diesem Anhang geht es darum, wie Sie beide Produkte kombinieren können.

Kombination auf Dateiebene. Die Kombination auf Dateiebene ist bei sehr großen PDF-Dokumenten empfehlenswert oder wenn Sie die Größe des Gesamtspeichers der PDFlib/PLOP DS-Kombination reduzieren müssen. Erzeugen Sie dazu einfach eine PDF-Datei mit den geeigneten PDFlib-Routinen auf der Festplatte und verarbeiten Sie das Dokument mit *PLOP_open_document( )*.

Erzeugen von Dokumenten im Speicher und digitales Signieren. Die speicherbasierte Methode ist zwar schneller, benötigt jedoch mehr Speicherplatz. Sofern es sich nicht um sehr große Dokumente handelt, empfehlen wir in webbasierten Anwendungen die folgende Methode für dynamische PDF-Erzeugung und -Signatur:

- Statt mit PDFlib eine PDF-Datei auf der Festplatte zu erzeugen, verwenden Sie die interne PDF-Erzeugung durch Übergabe eines leeren Dateinamens an *PDF_begin_document( )*.
- Holen Sie die erzeugten PDF-Daten ab durch den Aufruf von *PDF_get_buffer( )* nach *PDF_end_document( )*.
- Erzeugen Sie in PLOP eine auf den PDF-Daten im Speicher basierende virtuelle Datei durch einen Aufruf von *PLOP_create_pvf( )*.
- Übergeben Sie den Namen der PVF-Datei mit *PLOP_open_document ()* an PLOP DS.

Das Programmbeispiel *hellosign*, das in allen PLOP-Paketen enthalten ist, zeigt, wie PDFlib für die dynamische Erzeugung eines PDF-Dokuments sowie für die Übergabe an PLOP DS zum digitalen Signieren eingesetzt werden kann.

Dynamisches Erzeugen von Visualisierungsdokumenten. Mit PDFlib können Sie ein Dokument auch dynamisch erzeugen, dass zur Visualisierung einer Signatur verwendet wird (siehe Abschnitt 7.3.1, »Visualisieren von Signaturen mit Grafik oder Logo«, Seite 105). Dies ist nützlich, wenn Sie variable Text- oder Bild-Komponenten in das Visualisierungsdokument einbinden möchten, zum Beispiel das aktuelle Datum und die Uhrzeit.

Das Programmbeispiel *dynamicsign* zeigt, wie PDFlib für die dynamische Erzeugung eines PDF-Visualisierungsdokuments sowie für die Übergabe an PLOP DS zum digitalen Signieren eingesetzt werden kann.

Mit PDFlib erstellte Formularfelder . Wenn Acrobat ein Dokument mit Formularfeldern öffnet, erzeugt Acrobat die grafische Darstellung automatisch, falls erforderlich. PDFlib 7/8/9 stützt sich auf dieses Verhalten und erzeugt keine sogenannten Appearance-Streams. Da jedoch die automatische Erstellung der Appearance Streams in Acrobat das Dokument sofort nach dem Öffnen ändert, sind diese Dokumente für die digitale Signatur nicht geeignet.

PLOP DS lehnt daher solche Dokumente (beim Signieren im Update-Modus) standardmäßig ab. Im Rewrite-Modus, das heißt bei *update=false,* können solche Dokumen-

te durch Übergabe der Option *sacrifice={fields}* signiert werden. Allerdings werden Formularfelder aus dem Eingabedokument in der signierten Ausgabe nicht mehr vorhanden sein, wenn Sie diese Option verwenden.

C Kurzreferenz für die PLOP-Bibliothek

Die folgenden Tabellen geben einen Überblick über alle PLOP API-Funktionen. Das Präfix *(C)* bezeichnet C-Funktionsprototypen, die in der Java-Sprachbindung nicht zur Verfügung stehen.

Allgemeine Funktionen

Funktionsprototyp	*Seite*
*(C) PLOP*PLOP_new(void)*	*142*
void delete( )	*142*
void create_pvf(String filename, byte[ ] data, String optlist)	*142*
int delete_pvf(String filename)	*143*
double info_pvf(String filename, String keyword)	*144*

Dokumenteingabe und -ausgabe

Funktionsprototyp	*Seite*
int open_document(String filename, String optlist)	*145*
*(C) int PLOP_open_document_callback(PLOP *plop, void *opaque, size_t filesize, size_t (*readproc)(void *opaque, void *buffer, size_t size), int (*seekproc)(void *opaque, long offset), const char *optlist)*	*148*
int create_document(String filename, String optlist)	*149*
void close_document(int doc, String optlist)	*148*
byte[ ] get_buffer( )	*153*
int prepare_signature(String optlist)	*156*

Fehlerbehandlung

Funktionsprototyp	*Seite*
int get_errnum( )	*167*
String get_errmsg( )	*167*
String get_apiname( )	*167*

Globale Optionen

Funktionsprototyp	*Seite*
void set_option(String optlist)	*169*

pCOS-Funktionen

Funktionsprototyp	*Seite*
double pcos_get_number(int doc, String path)	*173*
string pcos_get_string(int doc, String path)	*173*
byte[ ] pcos_get_stream(int doc, String optlist, String path)	*174*

Funktion zur Unicode-Konvertierung

Funktionsprototyp	*Seite*
string convert_to_unicode(wstring inputformat, string input, wstring optlist)	*176*

D Änderungen

Änderungen an diesem Handbuch

Datum	Änderungen
01. März 2017	► Deutsche Übersetzung des Handbuch zu PLOP 5.2 und PLOP DS 5.2
25. Mai 2016	► Deutsche Übersetzung des Handbuch zu PLOP 5.1 und PLOP DS 5.1
18. März 2015	► Deutsche Übersetzung des Handbuch zu PLOP 5.0 und PLOP DS 5.0

Index

A

AATL 32, 93
Ad-Ticket-Schema 25
AES-Verschlüsselungsalgorithmus 62
Attributzertifikate 127
Aufgeben von Eigenschaften des Eingabe-PDFs 27
Authenticode-Zeitstempel 128
Authority Info Access (AIA) 117, 131
Autorensignaturen 113

B

Benutzerkennwort 61
Berechtigungskennwort 61
BES (Basic Electronic Signature) 134
beschädigte PDF-Eingabe 22
Brainpool-Kurven für ECDSA 104
Byteserving 20

C

C++ und .NET 53
C++-Sprachbindung 48
CAdES (CMS Advanced Electronic Signatures) 134, 160
CDS 94
cer Zertifikatformat 179
Certificate Revocation 90
Certificate Revocation Lists (CRLs) 120
CLI 48
CMS (Cryptographic Message Syntax) 75, 134
Commitment Type Indication 134
COM-Sprachbindung 50
critical-Flag im TSA-Zertifikat 127
CRL distribution point (CRLdp) 120
crt Zertifikatformat 179
C-Sprachbindung 45

D

Dateianlagen, verschlüsselt 64
DER-Kodierung 121
digitale IDs 89
digitale Signaturen 27, 89
Document Security Store (DSS) 111, 120, 135, 158
Dokumentebene
Zeitstempel 92, 125
Dokument-Infofelder 24
DSA-Signatur 104

E

ECDH (Elliptic Curve Diffie-Hellman-Schlüsselvereinbarungsschema) 78
ECDSA-Signaturen (elliptische Kurven) 104
eIDAS (Electronic identification and trust services) 95
elektronische Signaturen: siehe digitale Signaturen
Elliptic Curve Diffie-Hellman-Schlüsselvereinbarungsschema 78
EN 319 142-1 (Bausteine und PAdES-Basis-Signaturen) 135
Enveloped Data (CMS) 75
EPES (Explicit Policy-based Electronic Signature) 134
Erweiterung eines Signatur mit einem Archiv-Zeitstempel 138
ETSI EN 319 142-2 (Erweiterte PAdES-Signaturen) 135
ETSI EN 319 422 123
ETSI TS 101 733 (CAdES) 134
ETSI TS 102 778 (PAdES) 134
ETSI-Standards (European Telecommunications Standards Institute) 134
EUTL 32, 94
Evaluierungsversion 7

F

Fontoptimierung 21
Formularfelder im Eingabedokument 28

G

Garbage Collection 21
Genehmigungssignaturen 91
Ghent Workgroup (GWG) 25
große PDF-Dokumente 29

H

Hardware Security Module (HSM) 94, 99
Hash-Funktion für digitale Signaturen 104
HSM 99

I

id-pkix-ocsp-nocheck 119
inkrementelles PDF-Update 111
Installation PLOP/PLOP DS 7

J

Java-Sprachbindung 51

K

Kennwort für Dateianhänge 61
Kennwortdatei für digitale IDs 97
Kennwörter 61, 62
für digitale IDs 97
Unicode 63
kommerzielle Lizenz 11
kryptografische Engines 96
kryptografische Tokens 96, 97

L

Langzeitvalidierung (Long-Term Validation) 129, 134
LDAP 131
linearisiertes PDF 20
Lizenzschlüssel 9
Logging 171

M

Massensignaturen 99
Master-Berechtigung 76
Master-Kennwort 61
MDP-Signatur (Modification Detection and Prevention) 92
Message Digest für digitale Signaturen 104
Microsoft Cryptographic API (MSCAPI) 96, 100
Multithreading für PKCS#11 99

N

.NET-Sprachbindung 53
Nicht eingegrenzte String-Werte in Optionslisten 140
noaccessible 67
noannots 66
noassemble 67
no-check-Erweiterung (OCSP) 119
nocopy 67
noforms 67
nohiresprint 66
nomaster 67
nomodify 66
noprint 66

O

Object Identifier (OID) 180
Objective-C-Sprachbindung 54
OCSP (Online Certificate Status Protocol) 117
OCSP-Erweiterung no-check 119
optimiertes PDF 20
Optimierung 21
Optionslisten 139

P

PAdES (PDF Advanced Electronic Signatures) 134, 160
erweiterte Signaturprofile E-BES, E-EPES, E-LTV 135
Signaturstufen B-B, B-T, B-LT, B-LTA 135
Teile 134
pCOS
API-Funktionen 173
Cookbook 12
PDF mit Reader-Funktionalität 29
PDF/A 27, 28
und Signaturen 107
und XMP-Metadaten 25
PDF/UA 27
PDF/VT 27, 106
PDF/X 27, 28, 106
PDFlib mit PLOP/PLOP DS 181
PDF-Update 111
PDF-Version der generierten Ausgabe 27
PEM-Kodierung 121, 179
Perl-Sprachbindung 56
PFX-Format 97
PHP-Sprachbindung 57
PKCS#11 96, 97
PKCS#12 97
plainmetadata 67
PLOP/PLOP DS-Bibliothek
API-Referenz 139
Funktionen 17, 31
Kurzreferenz 183
PLOP/PLOP DS-Kommandozeilen-Tool
Beispiele 43
Funktionen 17, 31
Optionen 39
Rückgabewerte 42
PLOP_CATCH( ) 168
PLOP_close_document( ) 148
PLOP_convert_to_unicode( ) 176
PLOP_create_document( ) 149, 154
PLOP_create_pvf( ) 142
PLOP_delete( ) 142
PLOP_delete_pvf( ) 143
PLOP_EXIT_TRY( ) 46, 168
PLOP_get_apiname( ) 167
PLOP_get_buffer( ) 153
PLOP_get_errmsg( ) 167

PLOP_get_errnum( ) 167
PLOP_info_pvf() 144
PLOP_new( ) 142
PLOP_open_document( ) 145
PLOP_open_document_callback( ) 148
PLOP_pcos_get_number( ) 173
PLOP_pcos_get_stream( ) 174
PLOP_pcos_get_string( ) 173
PLOP_prepare_signature( ) 156
PLOP_RETHROW( ) 168
PLOP_set_option( ) 169
PLOP_TRY( ) 168
Protokollierung 171
Prüfung der Zertifikatsperrung 90
Python-Sprachbindung 59

R

RC4-Verschlüsselungsalgorithmus 62
Rechtecke in Optionslisten 140
Reparaturmodus für beschädigtes PDF 22
Response-Datei 42
RFC 2560 (OCSP) 117
RFC 2630 (CMS-Syntax) 127
RFC 3126 (signierte Attribute) 127
RFC 3161 (Zeitstempel) 123
RFC 3280
(Authority Info Access für calssuers) 131
(Authority Info Access für OCSP) 117
(CRL) 120
RFC 5126 (CAdES) 134
RFC 5480 (ECDSA mit NIST-Kurven) 104
RFC 5639 (ECDSA mit Brainpool-Kurven) 104
RFC 5652 75
RFC 5652 (CMS) 134
RFC 5816 (Zeitstempel) 123
RFC 6960 (OCSP) 117
Richtlinien-Identifikator 134
RPG-Sprachbindung 60
RSA-Signatur 104
Ruby-Sprachbindung 60
Rückgabewerte 42

S

SafeNet-Token 94
Schlüssellänge für digitale Signaturen 102
seitenweises Herunterladen 20
Session-Verarbeitung für PKCS#11 99
SHA-256 Message Digest 104
Signaturen mit eingebettetem Zeitstempel 124
Signaturen: siehe digitale Signaturen
Signaturtypen in PDF 91
Smartcards 96, 97
Stream-Optimierung 21
Strings in Optionslisten 140

T

Temporär erforderlicher Plattenspeicher 29
Thales nShield HSM 99
Time-Stamp Authority (TSA) 123
TimeStamp-Erweiterung 124

U

ungültige XMP-Metadaten 26
Unicode-fähige Sprachbindungen 141
Unicode-Passwörter 63
unnötige Objekte 21
Update 111

V

Verarbeitung von Exceptions 167
in C 45
verschlüsselte Dateianhänge 28, 64
Verschlüsselungsalgorithmus für digitale Signaturen 102
Verwendungsrechtesignaturen 93
Visualisieren von Signaturen 105

W

web-optimiertes PDF 20
Wörterbuchangriff 63

X

XMP-Metadaten 24, 25
Klartext 64
ungültig 26

Z

Zeitstempel 90
Zeitstempelsignaturen (Dokumentebene) 92, 125
Zertifikate 89
Zertifikatskette 89
Zertifikatsperrlisten 120
Zertifikatverwaltung unter Windows 101
Zertifizierungssignaturen 92, 113
Zugriffsberechtigungen 63

PDFlib GmbH

PDFlib GmbH
Franziska-Bilek-Weg 9
D-80339 München
www.pdflib.com
Tel. +49 • 89 • 452 33 84-0
Fax +49 • 89 • 452 33 84-99

Bei Fragen können Sie die PDFlib-Mailing-Liste abonnieren
und sich deren Archiv ansehen unter groups.yahoo.com/neo/groups/pdflib/info

Vertriebsinformationen
sales@pdflib.com

Support
support@pdflib.com *(geben Sie bitte immer Ihre Lizenznummer an)*