PDFlib GmbH

PLOP・PLOP DS

Version 5.4

PDF の線形化・最適化・保護・デジタル署名

PDFlib GmbH
Franziska-Bilek-Weg 9, 80339 München, Germany
www.pdflib.com
電話 *+49 • 89 • 452 33 84-0*

疑問がおありの際は、*groups.yahoo.com/neo/groups/pdflib/info* にある *PDFlib* メーリングリストをチェックしてください。

ライセンスご希望の際の連絡先：*jp.sales@pdflib.com*
商用 *PDFlib* ライセンス保持者向けサポート：*jp.support@pdflib.com*（ライセンス番号をお知らせください）

この出版物およびここに含まれた情報はありのままに供給されるものであり、通知なく変更されることがあり、また、*PDFlib GmbH* による誓約として解釈されるべきものではありません。*PDFlib GmbH* はいかなる誤りや不正確に対しても責任や負担を全く負うものではなく、この出版物に関するいかなる類の（明示的・暗示的または法定に関わらず）保障をも行うものではなく、そして、いかなるそしてすべての売買可能性の保障と、特定の目的に対する適合性と、サードパーティの権利の侵害とを明白に否認します。

PDFlib と *PDFlib* ロゴは *PDFlib GmbH* の登録商標です。*PDFlib* ライセンス保持者は *PDFlib* の名称とロゴを彼らの製品の文書内で用いる権利を与えられます。ただし、これは必須ではありません。

PDFlib PLOP 及び *PLOP DS* は以下のサードパーティソフトウェアの改変された部分を含んでいます：
ICClib、*Copyright © 1997-2002 Graeme W. Gill*
Zlib 圧縮ライブラリ、*Copyright © 1995-2017 Jean-loup Gailly and Mark Adler*
Eric Young の書いた *Cryptographic* ソフトウェア 、*Copyright © 1995-1998 Eric Young*（*eay@cryptsoft.com*）
Independent JPEG Group の *JPEG* ソフトウェア、*Copyright © 1991-2017, Thomas G. Lane, Guido Vollbeding*
OpenSSL Cryptographic Library, Copyright © 1998-2018 The OpenSSL Project（*www.openssl.org*）
Expat XML パーサ、*Copyright © 2001-2017 Expat* メンテナ
ICU International Components for Unicode、*Copyright © 1995-2012 International Business Machines Corporation* ら
Curl multiprotocol file transfer library、*Copyright © 1996-2018, Daniel Stenberg*（*daniel@haxx.se*）
OpenJPEG library, Copyright © 2012, CS Systemes d'Information, France

PDFlib PLOP 及び *PLOP DS* は *RSA Security, Inc.* の *MD5* メッセージダイジェストアルゴリズムを含んでいます。

目次

0 初めての PLOP・PLOP DS

0.1 ソフトウェアをインストール

PLOP と PLOP DS は、Windows システム版は統合インストーラパッケージとして、その他すべての対応オペレーティングシステム版は統合圧縮アーカイブとして頒布されています。インストーラとアーカイブの中に、PLOP/PLOP DS コマンドラインツールと PLOP／PLOP DS ライブラリが入っており、説明書と作成例も同梱されています。パッケージをインストールまたは解凍した後は、以下のステップを推奨します：

- さまざまな機能の概略紹介が 1 章「PLOP の諸機能」（17 ページ）にあります。
- PLOP/PLOP DS コマンドラインツールの利用者は、その実行形式をただちに使用することができます。利用できるオプションは 3.1 節「PLOP・PLOP DS コマンドラインオプション」（39 ページ）で説明されているほか、PLOP コマンドラインツールにオプションを何も付けずに実行したときにも表示されます。
- PLOP/PLOP DS ライブラリ／コンポーネントの利用者は、選んだ環境に応じて 4 章「PLOP・PLOP DS ライブラリの言語バインディング」（45 ページ）の中のいずれかの節を読み、インストールされている作成例に目を通すべきです。Windows では、PLOP と PLOP DS のプログラミング作成例は、スタートメニューから（.NET の場合）、あるいはインストレーションディレクトリから（それ以外の言語バインディングの場合）呼び出すことができます。

PLOP または PLOP DS の商用ライセンスを入手した場合は、次のページに従って、自分のライセンスキーを適用する必要があります。

評価版の制限　PLOP/PLOP DS のコマンドラインツールとライブラリは、商用ライセンスがなくても、完全動作する評価版として使用することができます。PLOP または PLOP DS の未ライセンス版は、業務目的に使用してはならず、その製品を評価するためにのみ使用することができます。その製品を業務環境に実装するには、有効なライセンスが必要です。

有効なライセンスキーを適用しないと、PLOP は、*unlicensed* というテキストを出力文書のメタデータに入れ込んで、文書の先頭に追加の表紙ページを挿入します。試験を実行するために、以下の条件の一方ないし両方が真ならば表紙は生成されません：

- 暗号化を、決まったパスワード文字列 *demo* または *DEMO* で行うとき（*userpassword*・*masterpassword* オプション）。
- 署名するために使用するデジタル ID の共通名（*CN*）が *demo* または *DEMO* を含むとき。試験に適したデジタル ID 群が PLOP DS パッケージに入っています。
- *subject* フィールド内の共通名（*CN*）が *demo* または *DEMO* を含む受信者証明書を用いて証明書セキュリティによって文書を保護するとき。試験に適した証明書群が PLOP DS パッケージに入っています。

場合によっては、入力が PDF/A・PDF/UA・PDF/VT・PDF/X のいずれかの規格に準拠していても、この表紙の挿入によって、PDF 出力がそれに準拠しなくなることがあります。この非準拠はこの表紙ページのみに特有であり、有効なライセンスキーを適用した後にはこの問題は起こりません。

pCOS の諸機能は、評価モードでは、小さな文書に限定されます（10 ページ未満かつ 1 MB 未満）。

plop.open_document( ) から取得された各文書ハンドルに対しては、評価モードでは、*plop.create_document( )* 呼び出しへのただ 1 回の呼び出しのみが許されます。

0.2 PLOP/PLOP DS ライセンスキーを適用

PLOP/PLOP DS を実用目的に使用するには、有効なライセンスが必要です。ライセンスを購入したら、追加表紙ページが出ないように、また任意のパスワードが使えるようにするために、自分のライセンスキーを適用する必要があります。ライセンスキーの適用にはいくつかの方法があります。以下に示す方法のいずれかを選んでください。

PLOP_set_option() の *frontpage* オプションを *false* にすると、有効なライセンスキーが見つからなかったときに表紙ページが生成されず例外が発生します。

注記 PLOP/PLOP DS ライセンスキーはプラットフォーム依存であり、その購入対象のプラットフォームでのみ利用できます。PLOP DS ライセンスキーでは PLOP の全機能が有効になりますが、PLOP ライセンスキーでは、PLOP DS でのみ利用できる署名機能は有効になりません。

注記 PLOPまたはPLOP DSのライセンスは、すべてのPLOPパッケージに内蔵されているpCOSコマンドラインツールをもカバーしています。

Windows インストーラ Windows ユーザーは、提供されているインストーラを使って PLOP ／ PLOP DS をインストールする際に、ライセンスキーを入力することができます。Windowsではこの方法を推奨します。レジストリへの書き込みアクセスを持たない場合や、インストーラを使えない場合は、以下に示す代替方式を参照してください。

API 呼び出しで実行時にライセンスキーを適用 動作時にライセンスキーを設定する行を、自分のスクリプトまたはプログラムに追加します。PLOP オブジェクトをインスタンス化した直後に（すなわち、*PLOP_new()* または同等の呼び出しの後に）*license* パラメータを設定する必要があります。具体的な文法は、使うプログラミング言語によります：

- C++・Java・.NET/C#・Python・Ruby の場合：

```
plop.set_option("license=...あなたのライセンスキー ...")
```

- C の場合：

```
PLOP_set_option(p, "license=...あなたのライセンスキー ...");
```

- Perl・PHP の場合：

```
$plop->set_option("license=...あなたのライセンスキー ...")
```

ライセンスファイルを使用 実行時呼び出しによってライセンスキーを与えるのではなく、テキストファイル内に以下の形式に従ってライセンスキーを書き込むこともできます（PLOP ディストリビューションに含まれているライセンスファイルテンプレート *licensekeys.txt* を使えます）。「#」キャラクタで始まる行はコメントを内容としますので無視されます。2 行目はライセンスファイル自体のバージョン情報を内容とします：

```
# PDFlib GmbH製品のライセンシング情報
PDFlib license file 1.0
PLOP        5.4             ...あなたのライセンスキー ...
```

ライセンスファイルには、複数の PDFlib GmbH 製品のライセンスキーを、個々の行ごとに含めることもできます。また、複数のプラットフォーム用のライセンスキーを含めて、1 個のライセンスファイルを複数のプラットフォームで使いまわすことも可能です。ライセンスファイルは以下の方法で設定できます：

- *licensekeys.txt* という名前のファイルが、すべてのデフォルト場所内で検索されます（「デフォルトファイル検索パス」（10 ページ）参照）。
- *licensefile* パラメータを *set_option()* API メソッドで設定することもできます：

```
plop.set_option("licensefile={/path/to/licensekeys.txt}");
```

- PLOP または pCOS コマンドラインツールの *--plopopt* オプションを用いて、*licensefile* オプションをライセンスファイルの名前とともに与えます：

```
plop --plopopt "licensefile /path/to/your/licensekeys.txt" ...
pcos --plopopt "licensefile /path/to/your/licensekeys.txt" ...
```

  パス名に空白キャラクタが含まれる場合には、パスを中括弧で囲う必要があります：

```
plop --plopopt "licensefile {/path/to/your/license file.txt}" ...
pcos --plopopt "licensefile {/path/to/your/license file.txt}" ...
```

- ライセンスファイルを指し示す環境（シェル）変数を設定することもできます。Windows では、システムコントロールパネルを用いて**「システム」→「詳細設定」→「環境変数」**を選択します。Unix では、下記のようなコマンドを適用します：

```
export PDFLIBLICENSEFILE=/path/to/licensekeys.txt
```

ライセンスキーをレジストリに Windows では、ライセンスファイルの名前を下記レジストリ値に書きこむこともできます：

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

あるいは、ライセンスキーを直接下記レジストリ値のいずれかに書きこむことも可能です：

```
HKLM\SOFTWARE\PDFlib\PLOP5\license
HKLM\SOFTWARE\PDFlib\PLOP5\5.0\license
```

インストーラはライセンスキーをこれらのエントリの末尾へ書き込みます。

注記 64 ビット Windows システム上で手作業でレジストリを操作する際には注意が必要です。通常、64 ビット PLOP バイナリは Windows レジストリの 64 ビットビューとともに動作するのに対して、64 ビットシステム上で走る 32 ビット PDFlib バイナリはレジストリの 32 ビットビューとともに動作します。32 ビット製品に対するレジストリキーを手作業で追加する必要がある場合には、必ず、*regedit* ツールの 32 ビットバージョンを使用してください。これは「スタート」ダイアログから下記のように呼び出すことができます：

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

デフォルトファイル検索パス Unix・Linux・macOS システムでは、ファイルに対してパス・ディレクトリ名を指定していなくても、いくつかのディレクトリがデフォルトで検索されます。以下のディレクトリが検索されます：

```
<rootpath>/PDFlib/PLOP/5.4/resource/cmap
<rootpath>/PDFlib/PLOP/5.4/resource/codelist
<rootpath>/PDFlib/PLOP/5.4/resource/glyphlst
<rootpath>/PDFlib/PLOP/5.4/resource/fonts
<rootpath>/PDFlib/PLOP/5.4/resource/icc
<rootpath>/PDFlib/PLOP/5.4
<rootpath>/PDFlib/PLOP
<rootpath>/PDFlib
```

Unix・Linux・macOS では、*<rootpath>* はまず */usr/local* へ置き換えられ、ついで HOME ディレクトリへ置き換えられます。

ライセンスファイルのデフォルトファイル名　デフォルトでは、デフォルト検索パスディレクトリ内で下記のファイル名が検索されます：

```
licensekeys.txt
```

この機能を利用すれば、環境変数や実行時オプションを設定せずにライセンスファイルを扱うことが可能になります。

0.3 説明書とサンプル群への案内

PLOP 用各種ミニサンプル　PLOP ディストリビューションは、すべての対応言語バインディングのためのシンプルなプログラミング例を含んでいます。これらは基本的な PLOP ライブラリプログラミング作業を演示しています：

- *encrypt* サンプルは、暗号化されていない PDF 文書を、ユーザー・マスターパスワードを用いて暗号化します。
- *certsec* サンプルは、1 個ないし複数の受信者証明書に対して PDF 文書を暗号化します。サンプル受信者証明書群がパッケージに含まれています。パッケージは、この暗号化された文書を PLOP を用いて復号したり Acrobat で開いたりするために必要となる照応するデジタル ID ファイル群も含んでいます。すべてのデジタル ID ファイル（*demo_recipient_1.p12* 等）に対するパスワードは *demo* です。
- *dumper* サンプルは、pCOS インタフェースを用いて、文書の一般特性群、暗号化に関する情報、署名ステータスに加え、文書情報と XMP メタデータを収集します。
- *insertxmp* サンプルは、ファイルから XMP メタデータを読み取り、その XMP を PDF 文書内に挿入します。試験用のサンプル XMP ファイル群が与えられています。

PLOP DS 用各種ミニサンプル　以下のミニサンプル群が PLOP DS で使えます：

- *sign* サンプルは、既存の PDF 文書に電子署名を行う方法を示します。
- *multisign* サンプルは、複数の PDF 文書に電子署名を行う方法を示し、PKCS#11 トークンのためのセッション処理を演示します。
- *hellosign* サンプルは、PDFlib を用いてメモリ内に動的に文書を生成して PLOP DS へ渡し、PLOP DS でそれに電子署名を行う方法を示します。この例は、PLOP パッケージには含まれていない PDFlib 製品を必要とします。ただし、PDFlib の無料評価版を当社 Web サイトから入手できます。
- *dynamicsign* サンプルは、手書き署名のパーソナライズされた画像等を含む署名視覚化文書を PDFlib を用いて動的に生成する方法を示します。このサンプルには PDFlib 製品も必要です。

この署名サンプル群は、パッケージ内にも含まれているデモ電子 ID 群を用いるように作られています。すべての電子 ID ファイル群（*demo_signer_rsa_2048.p12* 等）に対するパスワードは *demo* です。

PLOP コマンドラインツールへの各種サンプル呼び出し　PLOP コマンドラインツールでは、3.1 節「PLOP・PLOP DS コマンドラインオプション」（39 ページ）で解説するさまざまなオプションを使用することができます。この他に本説明書内のいくつかの章にも、PLOP コマンドラインツールのサンプル呼び出しが含まれています。

pCOS クックブック　pCOS クックブックは、PLOP・PLOP DS に内蔵されている pCOS インタフェースのためのコード断片の集合です。以下の URL にあります：
www.pdflib.com/pcos-cookbook。

pCOS インタフェースの詳細は、PLOP パッケージに含まれている pCOS パスリファレンスで解説されています。

pCOS コマンドラインツールの呼び出し例　すべてのPLOPパッケージに内蔵されている pCOS コマンドラインツールについては、別途のマニュアルで解説しており、そこにサンプル例もあります。

0.4 PLOP と PLOP DS の概要

PLOPには2種類があります:PLOP基本製品と、電子署名に対応した拡張版PLOP DSです。

PLOP の諸機能　PLOP では以下のような PDF 処理ができます：

- パスワードセキュリティ：PDF 文書を、ユーザーまたはマスターパスワード（あるいは両方）を用いて暗号化。PDF 暗号化を、その文書のマスターパスワードを知っている場合に除去。権限設定群（印刷やテキスト抽出の不許可等）を、その文書のマスターパスワードを知っている場合に追加または除去。
- 証明書セキュリティ:1 個ないし複数の受信者証明書に対して PDF 文書を暗号化。保護された PDF 文書を、然るべきデジタル ID を用いて復号。証明書セキュリティのための権限設定を適用または除去。
- PDF 文書を線形化することによって、PDF ファイルを Web サーバから取得する際のビューア体験を向上。
- PDF 文書のサイズを最適化するために、冗長なオブジェクトを削減。
- 破損した PDF 文書を修復。
- 内蔵の pCOS インタフェースを用いて、文書のセキュリティ状態（ユーザーまたはマスターパスワードを用いて暗号化されている）、権限設定群、文書メタデータ等多数の特性に関する情報をクエリ。
- 定義済みまたはカスタム文書情報項目群を挿入・取得。
- XMP メタデータを挿入・取得。

PLOP DS の諸機能　PLOP DS は、PLOP のすべての機能に加え、PDF 文書に電子署名を行う機能を実現します。この署名は、タイムスタンプ・長期検証・PAdES 署名に対応しています。2.1 節「PLOP DS のさまざまな署名機能」（31 ページ）で、PLOP DS の電子署名機能のまとめがあります。

さまざまな利点　PDFlib PLOP・PLOP DS は以下の利点を実現します：

- すべての PLOP・PLOP DS 操作は、PDF/A・PDF/UA・PDF/VT・PDF/X 規格に対応しています：入力がこれらのいずれかの規格に準拠していれば、出力は、可能であればその同じ規格に準拠することが保証されています。これが可能でない場合には（PDF/A 入力に対して暗号化が要求された等）、その操作は拒絶されるか、あるいは規格識別が除去されます。
- PLOP/PLOP DS は、PDF を読み取り、暗号化、署名、書き込みするために一切のサードパーティソフトウェアを必要としないスタンドアローンツールです。
- PLOP/PLOP DS は、技術的にも法的にもサーバ上に実装することが可能で、完全にスレッドセーフであり、メモリリークに対する検査済みです。PLOP は、ヘビーなサーバ用途のために構築されており、Web サーバ環境において、または大容量バッチ処理等のために使用できます。
- PLOP/PLOP DS は、多数のプラットフォーム上で、いくつかのプログラミング環境で利用可能です。
- さらなる柔軟性のために、PLOP/PLOP DS は、コマンドラインツールとしても、さまざまな開発言語のためのプログラミングライブラリ（コンポーネント）としても利用可能です。

PLOP/PLOP DS コマンドラインツールかライブラリか　PLOP/PLOP DS は、さまざまな開発言語のためのプログラミングライブラリ（コンポーネント）としても、バッチ操作のためのコマンドラインツールとしても利用可能です。どちらも同じ機能集合を実現します

が、それぞれ異なる実装タスクに適しています。ライブラリとコマンドラインツールのどちらを使うかについて、いくつかのガイドラインを示します：

- コマンドライン PLOP/PLOP DS ツールは、PDF 文書をバッチ処理するのに適しています。プログラミングを一切必要とせずに、それでいて強力なコマンドラインオプション群を提供しており、それらを用いて複雑なワークフローへそれを統合することが可能です。PLOP/PLOP DS コマンドラインツールは、ライブラリの使用に対応していない環境から呼び出すこともできます。
- PLOP/PLOP DS　プログラミングライブラリは、.NET・Java（サーブレットを含む）・PHP・プレーン C・C++ アプリケーション開発等、広く使われているさまざまな開発環境に良く統合します。

PLOP/PLOP DS ライセンスは、コマンドラインツールとライブラリの両方をカバーしています。

0.5 PLOP・PLOP DS の更新点

PLOP 5.1 における一般の変更点：

- 証明書セキュリティ：デジタル証明書によって識別される受信者の集合に対して文書を暗号化
- 証明書セキュリティを用いて暗号化されている文書の詳細を取得するための pCOS インタフェース 11
- 言語バインディングとプラットフォーム対応のアップデート
- 言語バインディング群とカーネルの中のさまざまなバグ修正と改良

PLOP DS 5.1 における署名関連の変更点：

- RFC 5816（*SigningCertificateV2/ESSCertIDv2*）に従ったタイムスタンプ処理の更新
- 署名を生成する際のファイルサイズと処理を最適化
- OCSP のためのタイミングオプション群
- 間接 CRL に対応
- CRL 取得をより堅牢に。予期しない HTTP ヘッダに対応等
- PKCS#11 エンジンにおいて特定のトークン機種群のふるまいを扱う回避策
- ハードウェアセキュリティモジュール（HSM）を用いる署名に対応
- デフォルトで PAdES/CAdES 署名を生成
- ハッシュ化・署名のための外部暗号化エンジンと連携するための PLOP DS のカスタムビルド
- PDF 処理におけるバグ修正。フォームフィールド名・XMP プロパティ等
- PDFlib を用いて動的署名描像を生成するコードサンプルを追加
- 外部暗号化ルーチンを、PKCS#11 インタフェースを通じて、動的読み込みなしで連携させるための新たなビルド構成

PLOP 5.2 における変更点：

- RSA 署名に対する PSS 符号化スキームに対応
- 証明書セキュリティ：RSA のための OAEP パディングスキームに対応

PLOP 5.3 における変更点：

- 言語バインディングとプラットフォームの対応を更新
- 将来の ISO 規格群への対応と、限定モードでフォームフィールドにアクセスするためのセキュリティモデルの改良とを伴う pCOS インタフェース 12
- pCOS コマンドラインツールを追加
- PKCS#11 を通じた RSA 署名に対する PSS 符号化スキームにも対応
- Amazon CloudHSM に対応

PLOP 5.4 における変更点：

- 各種プラットフォーム・各種言語バインディングで多数の改善
- サードパーティコードをアップデートして潜在的なセキュリティ脆弱性を解決
- ネットワークアクセスのためのプロキシサーバに対応
- DSS に対して別途の PDF 更新を生成しない。いくつかの検証ツールが「署名が文書全体を対象としていません」とエラーを出すので。
- CRL と OCSP に対して転送圧縮を可能にすることによりネットワーク伝送を加速
- CRL をダウンロードする際に構成が誤っている CA に対する回避策を追加
- 構造がおかしい PDF と XML メタデータのための回避策をさらに追加
- Alpine Linux に対応

1 PLOP の諸機能

注記 PLOP DS の電子署名のための諸機能については 2 章「PLOP DS の諸機能（電子署名）」（31 ページ）で解説しています。

1.1 パスワードセキュリティと権限

パスワードを用いて PDF 文書を暗号化・復号すること、および権限制限について、詳しくは 5 章「パスワードセキュリティ」（63 ページ）で説明しています。本節では概観と、手はじめのいくつかの例を示します。

セキュリティ設定をクエリ　pCOS プログラミングインタフェースを用いて、パスワードセキュリティを用いて保護されている PDF 文書のさまざまなセキュリティ設定をクエリすることができます。必要な関数呼び出しと引数を、すべての PLOP パッケージに入っている *dumper* ミニサンプル内で見ることができます。pCOS コマンドラインツールを使用すると、プログラミングをせずに PDF 文書から情報をクエリできます（1.6 節「pCOS を用いて文書情報をクエリ」（22 ページ）にある例を参照）。

パスワードを用いて文書を暗号化　文書を暗号化するには、*PLOP_create_document()* で *userpassword* オプションか *masterpassword* オプション（ないし両方）を指定します。ただし、ユーザーパスワードには必ずマスターパスワードが必要ですが、その逆は真ではありません。PDF 文書を暗号化するサンプルコードを、すべての PLOP パッケージに入っている *encrypt* サンプルで見ることができます。PLOP コマンドラインツールでこれらと等価なオプションは *--user* と *--master* です。

例：ユーザーパスワード *demo* とマスターパスワード *DEMO* を用いてファイルを暗号化：

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

権限制限を指定　権限制限を指定するには、*PLOP_create_document()* の *permissions* オプションでさまざまなキーワードを設定します（表 5.3（68 ページ）参照）。PLOP コマンドラインツールでこれと等価なオプションは *--permissions* です。ただし権限設定には必ずマスターパスワードが必要です。

例：マスターパスワード *DEMO* を用いて文書を暗号化し、かつ、文書の印刷と内容のコピーを不許可：

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

パスワード保護されている文書を復号　文書を復号するには、*PLOP_open_document()* の *password* オプションで適切なユーザーパスワードかマスターパスワードを指定します。PLOP コマンドラインツールでこれと等価なオプションは *--password* です。

例：マスターパスワード *DEMO* を用いて 1 個のファイルを復号。入力文書にアクセス制限が適用されていても、それらはすべて除去されます（出力は復号されるので）：

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

暗号化と復号のさらなる例が 5.3 節「コマンドラインでパスワードセキュリティを適用」（70 ページ）にあります。

1.2 証明書セキュリティ

証明書を用いて PDF 文書を暗号化・復号すること、および権限制限について、詳しくは 6 章「証明書セキュリティ」（73 ページ）で説明しています。本節では概観と、手はじめのいくつかの例を示します。

セキュリティ設定をクエリ　pCOS プログラミングインタフェースを用いて、証明書セキュリティを用いて保護されている PDF 文書のさまざまなセキュリティ設定をクエリすることができます。必要な関数呼び出しと引数を、すべての PLOP パッケージに入っている ***dumper*** ミニサンプル内で見ることができます。pCOS コマンドラインツールを使用すると、プログラミングをせずに PDF 文書から情報をクエリできます（1.6 節「pCOS を用いて文書情報をクエリ」（22 ページ）にある例を参照）。

証明書を用いて文書を暗号化　文書を暗号化するには、***PLOP_add_recipient()*** を用いて受信者証明書 ***certificate*** を指定します。PDF 文書を暗号化するサンプルコードを、すべての PLOP パッケージに入っている ***certsec*** サンプルで見ることができます。PLOP コマンドラインツールでこれと等価なオプションは ***--recipient*** です。

例：証明書を用いてファイルを暗号化：

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

権限制限を指定　ある受信者に対する権限制限を指定するには ***PLOP_add_recipient()*** の ***permissions*** オプションを指定します。

例：ある受信者のための文書を暗号化し、かつ、印刷とコピーが不許可になるようその権限を制限：

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

証明書セキュリティを用いて保護されている文書を復号　文書を復号するには、***PLOP_open_document()*** の ***digitalid*** オプションで適切な受信者 ID を指定します。PLOP コマンドラインツールでこれと等価なオプションは ***--inputopt*** です。

例：パスワード保護されている PKCS#12 ファイルの中で得られるデジタル ID を用いて 1 個のファイルを復号：

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

暗号化と復号のさらなる例が 6.5 節「コマンドラインで証明書セキュリティを適用」（87 ページ）にあります。

1.3 Web 最適化（線形化）PDF

PLOP は、PDF 文書に対して線形化という処理を施すことができます。そこから生まれる特性は、Acrobat では「***Web* 表示用に最適化**」と呼ばれています。線形化は、PDF ファイルの中のさまざまなオブジェクトを認識して、情報を付加し、それによって表示を高速化します。

線形化されていない PDF は、クライアントへまるごと転送される必要がありますが、線形化された PDF であれば、Web サーバは、バイトサービングという処理を用いてそれを 1 ページずつ転送することが可能になります。これによって、Acrobat（ブラウザのプラグインとして動作している）は、PDF 文書内の各部分を個別に取得できるようになります。その結果、ユーザーは、文書全体がサーバからダウンロード完了するまで待たずに、その文書の先頭ページの閲覧を始めることができます。これはユーザー体験の向上をもたらします。

ただし、Web サーバが PDF データをストリーム転送する先はブラウザであって、PLOP ではありません。逆に、PLOP が、バイトサービング可能な PDF ファイルを生成するのです。PDF のバイトサービングを活用するためには、以下のすべての要請が満たされる必要があります：

- PDF 文書が線形化されている必要があります。これは PLOP で実現できます。線形化は、暗号化または復号と同時に適用可能です。Acrobat では、ファイルが線形化されているかを調べるには、その文書のプロパティを見ます（「Web 表示用に最適化：はい」）。
- ユーザーが Acrobat をブラウザのプラグインとして使っていて、かつ PDF ビューアでページごとのダウンロードを有効にしている必要があります（Acrobat XI/DC：「**編集**」→「**環境設定**」→「**インターネット**」→「***Web* 表示用に最適化を許可**」）。これはデフォルトで有効になっています。

PDF ファイルが大きければ大きいほど（ページ数で計るにせよ MB で計るにせよ）、それを Web で送受信する際に享受できる線形化の恩恵は大きくなります。

線形化と暗号化／復号は併せて適用可能です。ただし、保護されているファイルを線形化するには、適切なマスターパスワードを与える必要があります（表 5.2 参照）。

線形化とファイルサイズ　線形化は、大きな PDF 文書の Web ベース表示の向上を目指すものですので、1 ページ文書に対してはあまり意味がありません（可能ですが）。しかし、Acrobat のバグによって、小さな線形化された文書は常に線形化文書として処理されるわけではありません。たとえば、Acrobat は 4 KB より小さなすべての文書を非線形化文書と見なします。

Acrobat は、2 GB より大きな PDF 文書についても、線形化されていると見なしません。

PLOP で PDF 文書を線形化　線形化処理を有効にするには、*PLOP_create_document()* で *linearize* オプションを指定します。

PLOP コマンドラインツールでこれと等価なオプションは ***--webopt*** です。例：ディレクトリ内のすべての PDF 文書（これらはいずれもパスワードが必要でないと前提）を線形化し、できたファイルをターゲットディレクトリ ***output*** へコピー。詳細度レベル 2 は、すべての入力・出力ファイルについてその処理時にその名前を印字します：

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

1.4 最適化（軽量化）

PDF 文書を処理する際に、PLOP は、他のさまざまな操作に加えて、ファイル最適化を施すこともできます：

- PLOP は、同一データの重複出現を検出し、1 個以外すべてのインスタンスを削除します。これは主にフォントや画像が対象となりますが、それ以外の種類のデータについても適用されることがあります（ICC プロファイル等や、あるいはページ全体でさえも、その内容がまるごと同一であれば）。埋め込まれているフォントや画像は、もし他のフォントや画像の内容がまったく同じデータであれば、削除されます。削除されたデータへの参照はすべて、そのフォントや画像の残されたインスタンスへの参照へ置き換えられます。たとえば、複数の PDF 文書を合成して一つの文書にした場合、もしそれらに同じフォントが埋め込まれていたならば、できあがった PDF の中には余分なフォントデータが含まれてしまう可能性があります。PLOP はこの冗長なフォントデータを削除して、そのフォントのデータを 1 つだけ残します。
- 使用されていないオブジェクトは、**ガベージコレクション**として知られる処理によって、PDF ファイルから削除されます。場合によっては（Acrobat の「**名前を付けて保存 ...**」／「**別名で保存 ...**」コマンドでなく「**保存**」コマンドが使用されていると）Acrobat は、変更情報をファイルに追加して、文書の以前の状態を残したままにしています。PLOP は、文書の古いバージョンに関連したオブジェクトをすべて削除します。

PLOP では、情報の喪失につながるような最適化の仕方（フォントの埋め込みをやめたり、画像をダウンサンプルしたり等）は一切行いません。入力とまったく同じ品質で文書を表示したり印刷したりするために必要な情報がすべて、出力内へ引き継がれます。

こんにち、冗長オブジェクトの問題のある PDF 文書はごく一部のみとなっていることから、この最適化処理はデフォルトでは無効となっています。

PLOP を用いて PDF 文書を最適化　最適化処理を有効にするには、*PLOP_create_document()* で *optimize=all* オプションを指定、あるいは、PLOP コマンドラインツールで *--outputopt* オプションを指定します。

例：PLOP コマンドラインツールで文書を最適化：

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

PLOP を用いて XMP メタデータを除去　アプリケーションによっては、PDF 出力に、あらゆる状況で必要となるわけではない大量の XMP メタデータを付けて生成するものがあります。極端な場合には、PDF ファイル全体のサイズのほとんどを XMP メタデータが占めていることすらあります。こうした場合には、望まない XMP 文書メタデータを、PLOP を用いて以下のように除去できます：

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

これによって、詳細なメタデータを失うかわりに PDF ファイルサイズを大幅に削減できる可能性があります。XMP 内の標準識別子群（PDF/A のためのもの等）が失われることに留意してください。

1.5 破損 PDF に対する修復モード

PLOP は、破損している PDF のための修復モードを実装しており、ある種の破損文書をも処理することが可能になっています。しかしまれに、PLOP が修復できずに拒否される破損 PDF 文書もあります。

PLOP を用いて PDF 文書を修復　修復モードは、破損している入力に PLOP が出会ったときに自動的に有効になります。しかし、*PLOP_open_document( )* の *repair=force* オプションを使って、文書を開く際に何も問題が起こらなかった場合にも修復モードを強制することもできます。PLOP コマンドラインツールでこれと等価なオプションは *--inputopt repair=force* です。修復モードを無効にするには *repair=none* を指定します。

例：PLOP コマンドラインツールを用いて文書の再構築を強制：

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

無効な XMP メタデータ　PLOP は、XMP メタデータ内のある種の問題を修復します。しかし問題によっては修復できないものもあります。たとえば XML メタデータが XML 解釈エラーを引き起こした場合には常に、その XMP は使用不能とされます。PLOP では、無効な XMP に出会った場合の処理動作を制御するための *xmppolicy* オプションを提供しています。詳しくは「無効な XMP メタデータの扱い」（26 ページ）を参照してください。

1.6 pCOS を用いて文書情報をクエリ

PLOP ライブラリに内蔵されている pCOS インタフェースを使えば、PDF 文書のさまざまな特性をクエリすることができます。pCOS インタフェースについて詳しくは pCOS パスリファレンスで解説しています。pCOS を用いて文書情報をクエリするサンプルコードを、すべての PLOP パッケージに入っているミニサンプル ***dumper*** で見ることができます。

pCOS コマンドラインツールを用いて一般的な文書情報をクエリ　pCOS コマンドラインツールは、一般的な暗号化情報やフォント名など PDF 文書に関する情報を表示します。サポートしているすべてのオプションのさらなる使用例と解説が、pCOS コマンドラインツールマニュアルにあります。出力例：

```
pcos PLOP-manual.pdf
```

このプログラム呼び出しの出力結果は以下のようになります：

```
          File name: PLOP-manual.pdf
          File size: 1166699
        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
```

pCOS コマンドラインツールを用いて特定の情報をクエリ　PCOS コマンドラインツールを使用して、文書から特定の pCOS パスの値を取得することもできます。

以下のコマンドは、すべての注釈（リンクなどの種別）と、その下位種別、文書内の移動先、参照先 URL、リンク長方形のページ上の座標をリストします。注釈キーのリストは、プログラムへただ 1 つの引数として与える必要があるので、ダブルクォーテーションで囲う必要があります：

```
pcos --extended annotation "Subtype destpage A/URI Rect" file.pdf
```

以下のコマンドは先頭ページ上の注釈の数を印字します：

```
pcos --pcospath "length:pages[0]/Annots" file.pdf
```

さらなる使用例が pCOS コマンドラインツールマニュアルにあります。

1.7 文書情報項目を挿入・読み取り

PDF では、文書に関する一般情報を持つ文書メタデータとして、2 つの種類を利用することができます：文書情報項目と XMP メタデータです。

文書情報項目とは、キーに文字列を関連づけたものであり、構造化されていない何らかの情報を保持します。定義済みの情報キーである ***Subject・Title・Author・Keywords*** が広く利用されていますが、他にも特定の目的のために任意のカスタムキーを定義することができます。文書情報項目は、古くてシンプルな種類の PDF メタデータであるということができます。

PLOP を使えば、新しい文書情報項目を追加したり、既存の情報項目の値を書き換えたりすることができます。定義済みの項目もカスタムの項目も設定可能です。入力文書の中に XMP 文書メタデータがあった場合は、メタデータの整合性を保つために、標準とカスタムの情報項目が自動的に XMP へ同期されます。

PLOP で文書情報項目を挿入　文書情報項目を設定するには、***PLOP_create_document()*** で ***docinfo*** オプションを指定します。

例：定義済み文書情報項目「***Subject***」と、カスタム情報項目「***Department***」を指定。なお、「***Product Manual***」を中カッコで囲ってスペースキャラクタを保護しています：

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

このオプションは、以下のように***--outputopt*** オプションから PLOP コマンドラインツールへ与えることも可能です：

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

PDF 2.0 では、文書情報辞書は廃止となっています。ですので ***PLOP_create_document()*** の ***emitdocinfo*** オプションが ***true*** である場合にのみ出力されます。文書情報辞書が生成されない場合であっても、文書情報項目群はなお XMP へ同期されます。

PLOP を用いて文書情報項目を読み取り　PLOP ライブラリに内蔵されている pCOS プログラミングインタフェースを使えば、PDF 文書から文書情報項目（キーと値）を読み取ることもできます。必要な関数呼び出しと引数は、すべての PLOP パッケージに入っているミニサンプル ***dumper*** で見ることができます。

pCOS コマンドラインツールを使用すると、プログラミングをせずに PDF 文書から情報をクエリできます（1.6 節「pCOS を用いて文書情報をクエリ」（22 ページ）にある例を参照）。

PDF/A における文書情報項目　PDF/A 規格では文書情報項目に対して特別な取り扱いが義務付けられていることに留意してください：

- PDF/A-1：標準文書情報項目　***Title・Author・Subject・Keywords・Creator・Producer・CreationDate・ModDate*** は文書 XMP メタデータと同期されている必要があります。PLOP はこの同期を自動的に実現します。
- PDF/A-2/3：文書情報項目は存在してもよいですが、PDF/A 準拠リーダーによって無視されなければなりません。それらが存在する場合には、文書 XMP と同期しているべきであり、これは PDF/A-1 の場合と同じく PLOP によって自動的に行われます。

1.8 XMP メタデータを挿入・読み取り・除去

XMP（*Extensible Metadata Platform* ＝拡張可能なメタデータプラットフォーム）は、さまざまな定義済みプロパティを持った XML フレームワークの一種です。その名前が暗示するように、XMP は、個々の要請を満たす目的で、カスタムの拡張スキーマを用いて拡張することもできるようになっています。XMP は文書情報項目よりもはるかに強力であり、また PDF/A 等さまざまな標準規格において必須とされています。多くの業界団体が、XMP に基づいた規格を、デジタルイメージングやプリプレスデータ交換等、さまざまな垂直アプリケーションのために策定しています。

XMP に関するより詳しい情報や、他の情報源へのリンクが PDFlib ウェブサイトにあります。

PLOP を使えば、PDF 文書に XMP メタデータを挿入したり、PDF から XMP を読み取ったりすることができます。挿入された XMP の検証も行われるので、生成される出力は必ず正しいことが保証されます。入力文書が PDF/A 標準規格に準拠している場合、ユーザーが与える XMP は、PDF/A で定められている XMP の諸規則に準拠していなければなりません。これらの規則（XMP 拡張スキーマの検証を含む）について PLOP は検査を行いますので、PDF/A 入力にユーザーから与えられた XMP を加えた結果が必ず準拠 PDF/A 出力になることが保証されます。

PLOP による XMP の挿入は、以下の状況をはじめとする多くの状況で利用することができます（カッコ内は、PLOP ディストリビューションに含まれているサンプル XMP ファイルの名前です）：

- XMP メタデータを PDF/A 文書に追加。PDF/A 規格で定義されている XMP 拡張スキーマにも対応しています（*machine_pdfa1.xmp*）。
- デジタル化されたレガシ文書のスキャン過程を記述した XMP メタデータを追加（*engineering.xmp*）。
- Ghent Workgroup（GWG）Ad Ticket スキームに従った XMP メタデータを追加（*gwg_ad_ticket.xmp*）。詳しくは *www.gwg.org/download/job-tickets/* を参照してください。
- 会社独自の XMP メタデータを追加（*acme.xmp*）。

PLOP を用いて XMP メタデータを挿入　メタデータを挿入するためには、有効な XMP メタデータを UTF-8 形式で持つファイルを作成する必要があります。XMP を挿入するには、*PLOP_create_document()* で *metadata* オプションを指定します。このオプションには、いくつかのサブオプションも用意されています。PDF 文書に XMP を挿入するサンプルコードを、すべての PLOP パッケージに入っているミニサンプル *insertxmp* で見ることができます。

例：*gwg_ad_ticket.xmp* というファイルから XMP メタデータを挿入して、XMP 2004 標準規格に照らしてその XMP を検証させる：

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

文書情報項目を XMP へ同期　*PLOP_create_document()* に対する *docinfo* オプションを用いて挿入された文書情報項目群は、自動的に XMP へ同期されます。これは XML の詳細を取り扱う必要なく XMP を生成する便利な方法を提供します。

標準文書情報項目群は、照応する標準 XMP プロパティ群へ転写されます。カスタム文書情報項目群は *pdfx* XMP スキーマ内に転写されます（このスキーマ名は PDF Extension に由来しており PDF/X 規格とは無関係）。なお、PDF/A モードではカスタム文書情報項

目は XMP へ転写されません。なぜなら PDF/A ではカスタムプロパティに対して拡張スキーマ記述が必須だからです。

PLOP を用いて XMP メタデータを読み取り　PLOP ライブラリに内蔵されている pCOS プログラミングインタフェースを使えば、PDF 文書から XMP メタデータを抽出することもできます。必要な関数呼び出しと引数はミニサンプル ***dumper*** で見ることができます。ただし、このサンプル ***dumper*** は、実際に XMP メタデータを印字しているのではなく、単に文書内で見つかった XMP のサイズを報告しているだけです。

pCOS コマンドラインツールを用いて XMP メタデータを抽出することもできます。

PLOP を用いて XMP メタデータを除去　場合によっては、XMP メタデータを除去したい場合もあるでしょう。たとえば、それがもはや実際の文書内容に合致していない等の理由が挙げられます。これは PLOP で以下のように実現できます：

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

XMP メタデータを除去する際には標準識別子群（PDF/A のためのもの等）が失われることに留意してください。

無効な XMP メタデータの扱い　PDF 文書はときに、XML レベルで無効な、あるいは XMP/RDF レベルで無効な XMP メタデータを含んでいることがあります。PLOP はデフォルトでそのような文書を拒絶し処理を停止します。このような入力文書についてより細かい制御を行いたい場合は、***PLOP_open_document( )*** に対して ***xmppolicy*** オプションを用いれば、以下の場合を区別することができます：

- ***xmppolicy=rejectinvalid***：デフォルトでは、無効な XMP があれば PLOP は PDF 出力を生成しません。
- ***xmppolicy=ignoreinvalid***：無効な XMP を無視し、デバッグ支援のために生成出力 XMP 内に XMP 解釈エラーメッセージのテキストを含めます。このオプションでは PDF/A または PDF/X-3/4/5 出力は一切生成されないことに留意してください。
- ***xmppolicy=remove***：これは、望ましくないメタデータを削除するために役立ちます。

たとえば、無効な XMP メタデータによって文書群のバッチ処理が中断されるのを防ぐには、入力文書内の無効な XMP が引き起こす問題を無視することができます：

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

1.9 PLOP の処理の詳細

受け入れ可能な入力文書　PLOP は、以下の種類の PDF を受け入れ可能です：

- PDF 1.6（Acrobat 7）およびそれ以前の全バージョン
- PDF 1.7（Acrobat 8）。技術的に ISO 32000-1 と同等
- PDF 1.7 Adobe 拡張レベル 3（Acrobat 9）
- PDF 1.7 Adobe 拡張レベル 8（Acrobat X およびそれ以降）
- PDF 2.0。ISO 32000-2 に従ったものです

行いたい操作によっては、暗号化文書に対してパスワードが必要になる場合があります。PLOP は、さまざまな種類の破損 PDF 文書の修復を試みます。

PDF のバージョン　生成される出力文書の PDF バージョン番号は、入力文書の PDF バージョン番号よりも小さくなることは決してありませんが、強制的に高い番号へ上げさせられることがあります。PLOP は入力文書の PDF バージョンを使いますが、それは以下の規則に従って変更されます：

- PDF/A-1・PDF/X モードでは、PDF バージョンは変更されずに保たれます。PDF/A-2/3 モードでは、PDF 1.7 が生成されます。
- それ以外の場合には、PDF 出力バージョンは少なくとも PDF 1.6 です。
- パスワードセキュリティ（オプション *masterpassword*）は、PDF バージョンを、暗号化アルゴリズム 4 の場合には PDF 1.7ext3 へ上げ、暗号化アルゴリズム 11 の場合には PDF 1.7ext8 へ上げます（「パスワードセキュリティのための暗号アルゴリズムと鍵長」（67 ページ）を参照）。
- 証明書セキュリティ（関数 *PLOP_add_recipient()*）は、PDF バージョンを、pCOS アルゴリズム 6 の場合には PDF 1.6 へ上げ、pCOS アルゴリズム 10 の場合には PDF 1.7ext3 へ上げます（「PDF 暗号化アルゴリズムと鍵長」（83 ページ）を参照）。
- 署名機能のなかには、PDF バージョンを PDF 1.7ext8 へ上げるものがあります（表 7.1 参照）。

規格準拠　PLOP の処理はいくつかの PDF 規格に準拠しています。入力が以下のいずれかの規格に準拠している場合には、PLOP によって生成される出力はそれと同じ規格に準拠することが保証されます：

- PDF/A-1/2/3：すべての種類
- PDF/X-3/4/5・PDF/VT-1：すべての種類
- PDF/UA-1

PLOP の操作（とりわけ暗号化）のなかには、特定の規格と互換でないものもあることに留意してください。この場合には、*sacrifice* オプションを用いて優先順位を設定することもできます（以下を参照）。

入力 PDF の特定の特性を放棄　PDF 文書の特性のうちのいくつかは、特定の PLOP のアクションと衝突する可能性があります。たとえば、PDF/A 文書では暗号化を使うことは許されません。PDF/A 文書に暗号化をかけるよう指示されたら、PLOP はどのようにするべきでしょうか。デフォルトでは PLOP は、その操作を拒絶して例外を発生させます。しかし、*PLOP_create_document()* で、または PLOP コマンドラインツールの *--outputopt* オプションで、オプション *sacrifice* を用いることによって、行わせたいアクションに対して、入力特性よりも高い優先順位を与えることもできます。上記の例でいえば、暗号化を許すために、PDF/A 準拠項目は文書から除去されます。

入力文書の特性と、行わせたいアクションとの組み合わせは、いく通りかあります。そのいずれの組み合わせにおいても、*sacrifice* オプションを使えば、文書のある特定の特性を放棄することによって操作が許されます（詳しくは表 8.4 参照）：

- PDF/A：PLOP は電子署名を、PDF/A 準拠なやり方で適用します。PDF/A-1・PDF/A-2・PDF/A-3 のいずれかの標準規格に準拠している入力文書は、PDF/A 準拠の署名付き出力を生成することが保証されています。しかし暗号化は PDF/A 文書に対しては許されません。この規格では暗号化が一切禁止されているからです。しかし *sacrifice={pdfa}* オプションを指定することによって PDF/A 準拠を放棄することもできます。署名視覚化のために用いられている PDF ページも PDF/A も準拠する必要があります（7.3.1 節「グラフィックかロゴを用いて署名を視覚化」（105 ページ）参照）。
- PDF/X：PDF/X-1a/3/4/5 では、暗号化や、ページ上に可視の署名フィールドを置くことは許されていません。こうした状況では PLOP は例外を発生させますが、*sacrifice={pdfx}* オプションを指定することによって PDF/X 準拠を放棄することもできます。署名視覚化は PDF/X モードでは使えません。
- PDF/UA：多くの PLOP 操作は、*permissions=noaccessible* を例外として、PDF/UA-1 に準拠します。*sacrifice={pdfua}* オプションを用いて PDF/UA 準拠を放棄することもできます。
- PLOP は、視覚表現を持たない署名なしフォームフィールド（PDFlib 9.2 以下で作成されたフォームフィールド等）を持った文書には署名を適用できず、その種の入力に対してはエラーを発します。その理由は、見つからないフォームフィールド表現を文書内へ Acrobat は書き込むので、するとただちに署名は無効になってしまうためです。この場合、*PLOP_create_document()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションで、オプション *sacrifice={fields}* を指定することによって、既存フォームフィールド群を放棄することもできます。なお、このフォームフィールドに関する制約は、生成された署名を保持する署名フィールドに対してはあてはまりません。PDFlib 9.3 以上を用いて生成されたフォームフィールド文書には制約は一切ありません。
- 暗号化されていない文書の中に、暗号化されたファイル添付が入っているとき、そのパスワードが得られないと、処理はデフォルトで停止します。*PLOP_create_document()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションで、オプション *sacrifice={encryptedattachments}* を指定することによって、暗号化されたファイル添付群を放棄することもできます。このオプションを指定すると、パスワードが得られない暗号化されたファイル添付はすべて除去されます。
- 入力文書が 1 個ないし複数の電子署名を含んでおり、かつ更新モードで新規署名が一切作成されない場合には、処理はデフォルトで例外を出して停止します。*PLOP_create_document()* に対して、または PLOP コマンドラインツールの *--outputopt* オプションで、オプション *sacrifice={signatures}* を指定することによって、既存の署名群を放棄することもできます。

入力文書から無条件に失われる特性　以下の入力文書の特性は、PLOP のいかなる操作を施しても失われます：

- 入力文書が線形化されているとき、その線形化はデフォルトでは失われます。出力を線形化するには、*PLOP_create_document()* に *linearize* オプションを、または PLOP コマンドラインツールに *--linearize* オプションを与えます。なお、線形化は電子署名と併用することはできません。

- Reader 有効化された文書：Reader 有効化されている PDF 文書を PLOP で処理すると、Reader 有効化されていない出力が生成されます。Reader 有効化された PDF を作れるのは Adobe ソフトウェアだけですので、どうにかする方法はありません。

必要な一時ディスク容量　PLOP は入力 PDF 文書を読み込んで、出力 PDF を書き出します。出力文書は、おおよそ入力文書と同じディスク容量を必要とします（PLOP の最適化処理によって冗長な情報が削除されなければ）。多くの場合、これより多くのディスク容量が必要になることはありません。しかし PLOP/PLOP DS は、線形化か電子署名が有効にされているときには、その操作のために追加の一時ディスク容量を必要とします。

一時ファイルはデフォルトではカレントディレクトリに作成されますが、これは *PLOP_create_document()* の *tempdirname* オプションで変えることもできます。一時データのディスク容量は、おおよそ入力ファイルのサイズに等しくなります。線形化とインコア PDF 生成（すなわち出力ファイル名を与えない）をともに行うときは、PLOP は、おおよそ入力のサイズの 2 倍の一時ディスク容量を必要とします。

大容量 PDF 文書　多くのユーザーはギガバイト単位の PDF 文書を扱う必要には迫られないでしょうが、業務アプリケーションのなかには、大量の請求書や明細などを含む文書を作成したり処理したりする必要があるものがあります。PLOP 自体は生成する文書のサイズにいかなる制約も設けていませんが、PDF Reference やいくつかの PDF 規格によって課せられるいくつかの制限があります：

- 2 GB ファイルサイズ制限:PDF/A などの規格では、ファイルサイズを 2 GB までに制限しています。一文書がこの制限よりも大きくなる場合には、PLOP は PDF/A・PDF/X-4・PDF/X-5 出力を生成しているときには例外を発生させます。それ以外の場合であれば 2 GB を超える文書を作成できます。
- 10 GB ファイルサイズ制限:PDF 文書内の昔ながらの相互参照テーブルは、10 進 10 桁すなわち 10^{10}-1 バイトまでに制限されています。これはおよそ 9.3 GB にあたります。しかし、圧縮されたオブジェクトストリームを用いれば、この制約を超えることが可能です。圧縮されたオブジェクトストリームはいずれにせよ全体のファイルサイズを削減しますが、*objectstreams* 実装の一部である圧縮された相互参照ストリームはもはや 10 進 10 桁の制約に縛られず、それゆえ 10 GB を超える PDF 文書の作成を許容します。
- オブジェクトの数：一文書内のオブジェクトの数は全般的には PDF によって制限されていませんが、PDF/A・PDF/X-4・PDF/X-5 規格では、一文書内の間接オブジェクトの数を 8,388,607 個までに制限しています。一文書がこの制限を超えるオブジェクトを必要とするときは、PLOP は PDF/A・PDF/X-4・PDF/X-5 出力を生成しているときには例外を発生させます。他のモードでは、もっとオブジェクトの多い文書も必ず作成できます。このチェックは、オプション *limitcheck=false* を用いて無効にすることも可能です。

マルチスレッドプログラミング　PLOP 自体はシングルスレッドですが、マルチスレッドアプリケーションで安全に使用できます。よくある、1 つの PLOP オブジェクトが 1 つのスレッドでのみ使用される状況においては、マルチスレッドに関する注意は特に必要ありません。同一のオブジェクトが複数のスレッドで使用される場合には、アプリケーションは、この PLOP オブジェクトが複数のスレッドから同時にアクセスされることのないよう、スレッド間の同期をとる必要があります。典型的なシナリオとしては、PLOP オブジェクト群のプールを持っておき、各スレッドは、新規の PLOP オブジェクトを作成するのではなくこのプールから既存の PLOP オブジェクト 1 個を取得して、そして文書を生成してからそのオブジェクトが必要なくなった時点でそれをプールへ戻す、といった使い方

が考えられます。同一の PLOP オブジェクトを、その出力文書が完成する前に別のスレッドで使用するのは、めったに利点がなく、推奨されません。

PLOP でできないこと　以下の制約に留意してください：

- PLOP はクラッカーツールではありません。これを用いて、保護された文書に対するアクセスを、適切なマスターパスワードを知ることなく得ることはできません。
- 動的 XFA フォーム（Adobe Experience Manager（AEM）フォームとも呼ばれます）を処理することはできません。なぜならそれは純正 PDF 文書ではなく、薄い PDF レイヤー内にパッケージされた XML フォームだからです。

2 PLOP DS の諸機能（電子署名）

注記 PDF 文書に電子的に署名する機能は PLOP DS でのみ利用可能であり、PLOP 基本製品では利用できません。

PDF 文書に対する電子署名については、詳しくは 7 章「PLOP DS による電子署名」（89 ページ）で網羅します。この章では、出発点として、概要と、最初の例を提供します。

2.1 PLOP DS のさまざまな署名機能

PDF 署名のさまざまな特性

- 既存の PDF 署名フィールド内に署名を作成、もしくは署名を保持する新規のフィールドを生成。この署名は、ページ上の特定の位置において不可視にすることも可視にすることも可能です。
- ロゴや手書き署名のスキャン等の表現を PDF ページとして取り込むことによって電子署名を視覚化。
- 署名を破壊することなくフォーム記入等の文書変更ができるよう許可する PDF 認証（作成者）署名を作成。
- 検証情報を、ISO 32000-1 に従って署名内に直接格納することもできますし、ISO 32000-2 と PAdES パート 4 で仕様化されているように文書セキュリティストア（DSS）内に格納することもできます。
- 署名を、増分的な PDF 更新セクション内に行うことによって既存の署名群と文書構造を温存することもできますし、最適化・暗号化のために文書構造を書き換えることによって行うこともできます。

さまざまな PDF バージョン・規格 PLOP DS は、あらゆる標準的な PDF のバージョンと規格に対応しています：

- PLOP DS は、Acrobat DC すなわち PDF1.7（ISO 32000-1）拡張レベル 8 までのすべての PDF バージョンを処理します。PLOP DS は、PDF 2.0（ISO 32000-2）に準拠した文書を処理することもできます。
- PLOP DS は、PDF/A-1/2/3（ISO 19005）アーカイビング規格群に対応しています：入力文書が PDF/A に準拠していれば出力文書も準拠が保証されます。PLOP DS は、PDF/A に要求される XMP 拡張スキーマに完全対応しています。PDF/A 準拠の XMP メタデータを PDF 文書に挿入できる点は PLOP の重要な特長です。
- 同様に PLOP DS は、PDF/X-1a/3/4/5（ISO 15930）印刷業務規格群と、トランザクション印刷のための PDF/VT-1（ISO 16612-2）と、アクセシブル PDF のための PDF/UA-1（ISO 14289）に対応しています。

さまざまな署名規格

- PDF 1.7（ISO 32000-1）に従った CMS ベースの PDF 署名
- PDF 2.0（ISO 32000-2）に従った長期検証（LTV）のための署名
- 適格 eIDAS 署名のための ETSI TS 102 778 パート 2・3・4、ETSI EN 319 142 に従った PAdES（PDF 高度電子署名）、CAdES（ETSI TS 101 733）

さまざまな PAdES 準拠レベル

- 基本署名（PAdES Level B-B）

- 時刻付き署名（PAdES Level B-T）
- 長期検証資料付き署名（PAdES Level B-LT）
- 検証資料の長期可用性・完全性を実現する署名（PAdES Level B-LTA）：eIDAS 準拠のためには必須
- PAdES パート 3 に従った基本電子署名（PAdES Level E-BES）・明示的ポリシーベース電子署名（PAdES Level E-EPES）

タイムスタンプ

- RFC 3161・RFC 5816・ETSI EN 319 422 に従って、信頼された時刻認証局（TSA）からタイムスタンプを取得し、生成する署名内に埋め込み。TSA の詳細を AATL 証明書から読み取って構成を何ら要せずタイムスタンプを作成することも可能です。
- ISO 32000-2 と PAdES パート 4 に従って文書レベルタイムスタンプ署名を作成。文書レベルタイムスタンプは、個人署名を行うことなく文書の状態を保証します。

暗号署名の詳細

- RSA・DSA アルゴリズムに加え、楕円曲線暗号に基づく楕円曲線電子署名アルゴリズム（ECDSA）に従った署名。RSA については、PKCS#1 v1.5・PCKS#1 v2.1（PSS）に対応しています。
- 強固な署名・ハッシュ関数。
- 生成される署名内に完全な証明書チェーンを埋め込み。したがって、Adobe 認定信頼リスト（AATL）か欧州連合信頼リスト（EUTL）に載っている CA（認証局）からの証明書を持った署名を、クライアント側で何ら構成を必要とせず Acrobat・Adobe Reader 内で検証できます。
- オンライン証明書状態プロトコル（OCSP.。RFC 2560・RFC 6960 に拠る）応答と証明書失効リスト（CRL。RFC 3280 に拠る）を長期検証（LTV）のための失効情報として埋め込み。

さまざまな署名エンジン　PLOP DS は、複数の暗号化エンジンに、すなわち電子署名を生成するためのコンポーネントに対応しています：

- 内蔵のエンジンは , 必要な暗号化機能を PLOP DS 内に、一切の外部依存なく実装しています。この内蔵エンジンは、PKCS#12・PFX 形式のソフトウェアベースのデジタル ID に対応しています。
- PLOP DS は暗号化トークンを、標準の PKCS#11 インタフェースを通じて紐付けることができます。この方法で、スマートカード・USB スティック等セキュアデバイス上のデジタル ID を用いて署名を行うことが可能です。セキュアな PIN 入力のためのキーボードが付いた機器でも同様です。
- PKCS#11 インタフェースを使用すると、ハードウェアセキュリティモジュール（HSM）を用いて署名を行うことも可能です。HSM は、セキュアな鍵ストレージを実現するとともに、大容量署名用途においてゆとりあるパフォーマンスを実現します。PLOP DS は、PKCS#11 セッションを使用することによって、HSM を用いたバルク署名のパフォーマンスを最大化します。PLOP DS は、AWS CloudHSM 等クラウドの HSM を用いて使用することも可能です。
- Windows では PLOP DS は、このオペレーティングシステム が提供している暗号化インフラストラクチャ（MS CAPI）を活用することができます。ソフトウェアベースのデジタル ID やセキュアハードウェアトークンに加えて Windows 証明書ストアからのデジタル ID を用いて署名を行うことが可能です。ただし LTV 等 MSCAPI エンジンでは利用できない署名機能もあります。

- あるいは、専用の暗号化ライブラリ内ですべての暗号化操作（ハッシュ化および署名）が実行されるようにするために、ユーザーが与える暗号化エンジンを使用することも可能です。

PLOP DS でできないこと　以下の制約に留意してください：

- PLOP DS を用いて PDF 文書を Reader 有効化する（Adobe Reader での注釈作成を許す等）ことはできません。なぜならこれには特定の Adobe 署名が必要だからです。
- 静的または動的 XFA フォームに署名を行うことはできません。

2.2 PLOP DS の評価のための準備

PDFlib デモ CA 証明書を Acrobat にインストール　以下の手順は、PLOP DS を用いて電子書名を作成するために必須ではありません。しかし PLOP DS を、そのパッケージ内で提供されているサンプル証明書群を用いて評価しようとするならば、以下の説明のように Acrobat を構成することを推奨します。これは、Acrobat の信頼済み証明書のリスト（「Acrobat における信頼済みルート証明書」（93 ページ）参照）にインストールされている商用 CA からの証明書で作業する場合には必要ありません。

PLOP DS に含まれているサンプル証明書群は PDFlib Demo CA によって発行・署名されています。この CA の自己署名ルート証明書を Acrobat で利用可能にすれば、生成される署名群は Acrobat 内で完全に有効として受け入れられます。PDFlib Demo CA 証明書を Acrobat XI/DC にインストールするには以下のように操作します：

- **「編集」→「環境設定」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」→「追加」→「参照 ...」**をクリック
- ***bind/data/PDFlibDemoCA_G3.crt***（PLOP インストレーションの一部）へブラウズし、**「追加」→「*OK*」**をクリック。
- これで、信頼済み証明書のリストの中にエントリ**「*PDFlib GmbH Demo CA G3*」**が現れます。このエントリを選択し、**「信頼を編集」**をクリックして、ボタン**「この証明書を信頼済みのルートとして使用」**・**「証明済み文書」**を有効にし、**「*OK*」**をクリックします。

デモデジタル ID を Windows に取り込む　Windows 上で PLOP DS の MSCAPI ベースの署名エンジンを試すには、Windows 証明書ストアでデジタル ID を利用可能にする必要があります。デモデジタル ID を取り込むには、照応する ***.p12*** ファイルをダブルクリックして「証明書のインポートウィザード」を起動し、その画面に従います。

2.3 PLOP DS で文書に署名

署名を行うにはデジタル ID が必要です。デジタル ID は、ファイルとして、または Windows 証明書ストア内で、あるいは暗号トークン（スマートカードや USB スティック等）上で利用できます。ファイルの場合には、そのデジタル ID にアクセスするにはパスワードが必要ですが、Windows 証明書ストアは通常、Windows ログインによって保護されており、パスワードを必要としません。暗号トークンは多くの場合、PIN によって保護されており、その PIN は、署名側ソフトウェアによって、あるいはそのトークンの内蔵キーボード上で直接、与えられる必要があります。

電子署名を作成するには ***PLOP_prepare_signature()*** を用います。いくつかのオプションを使えます。その後、その署名を行うには ***PLOP_create_document()*** を用います。PDF 文書に署名するためのサンプルコードが、すべての PLOP パッケージに含まれている ***sign*** ・

multisign ミニサンプル内にあります。PLOP コマンドラインツールでこれと同等のオプションは*--signopt* です。

基本的な署名オプションリストの例 PDF文書に対して、ファイル*demo_signer_rsa_2048.p12* からのデジタル ID を用いて、不可視署名を作成します。このデジタル ID に対するパスワード *demo* がファイル *pw.txt* の内容となっています：

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

（Windows のみ）PDF 文書に対して、Windows 証明書ストアからの（デフォルトストア *My* からの）証明書を用いて、不可視署名を作成します。これは、このデジタル ID は自分の Windows ログインによって保護されているのでパスワードを与える必要がないと前提しています：

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

PDF 文書に対して、暗号トークンからのデジタル ID を用いて、不可視署名を作成します。このトークンに対する PKCS#11 インタフェースは、そのスマートカードサプライヤーによって提供される必要のある *cryptoki.dll* ライブラリ内に実装されています。このデジタル ID に対するパスワードはファイル *pw.txt* 内に含まれています：

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

さらに詳しくは 7.2 節「PLOP DS を用いて署名する」（95 ページ）をごらんください。

2.4 証明用署名

証明用または作成者署名は、その文書をその作成者が作成した時点におけるその状態を証明すると同時に、その証明を破ることなくある種の変更を許すものです。*certification* オプションは、フォーム記入許可等、その証明された文書に対してその署名を破ることなく行うことのできる変更群を指定します：

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

2.5 タイムスタンプ

署名にタイムスタンプを追加するには、時刻認証局のURLが必要であり、それを*timestamp* オプションに与える必要があります：

```
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
```

同様に、文書レベルタイムスタンプを、*doctimestamp* オプションを用いて適用できます：

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

さらに詳しくは 7.5 節「タイムスタンプ」（121 ページ）をごらんください。

2.6 LTV 有効化署名

長期検証（long-term validation ＝ LTV）への対応には、チェーン内のすべての証明書が利用可能であり、かつ、証明書失効情報が署名の作成時にオンラインかディスクファイルから取得できることが必須です。このためには、然るべき OCSP または CRL サーバ群が PKI によって提供されている必要があります。多くの場合（とりわけ AATL 証明書）、必要なネットワーク情報は署名証明書から読み取ることが可能です。そうでない場合には、然るべきネットワークリソースを、*ocsp* および／または *crl/crlfile/crldir* オプションから与える必要があります。証明書チェーン全体へのアクセスを与えるために、そのルート CA 証明書を内容とする PEM ファイルの名前をオプション *rootcertfile* で与える必要があります。

LTV 有効化署名は通常、用いられている証明書群に対するオンライン PKI リソース群（CRL または OCSP）を必要とします。これは PLOP DS デモ証明書に対しては利用できません。次善策として、ディストリビューション内で提供されている CRL ファイル *PDFlibDemoCA_G3.crl* を利用できます（この CRL の失効日は、業務環境であれば受け入れがたいような非常に遠い未来になっています）。これに照応する、LTV 有効化署名を作成するためのコマンドライン呼び出しは、以下のようになります：

```
plop --signopt "digitalid={filename=demo_signer_rsa_2048.p12} password=demo ltv=full ↵
        crlfile=PDFlibDemoCA_G3.crl rootcertfile=PDFlibDemoCA_G3.pem" ↵
        --outfile ltv-signed.pdf input.pdf
```

次の例では、必要な OCSP または CRL 取得情報は署名証明書内に存在していると前提しています。商用証明書では通常そのようになっています。これらの条件下においては、オプション *ltv=full* を与えることにより、必ず LTV 有効化署名が作成されるようにすることができます：

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

関与する PKI の内容によってはこれでは充分でない可能性があることに留意してください。特に、失効情報は、OCSP/CRL 証明者と時刻認証局に対しても利用可能となっている必要があります。

2.7 PAdES 署名

各種 PAdES 署名は、PDF 署名を改良して、EU の諸要請を満たすようにしたものです。さまざまな署名オプションを用いて、さまざまな種類の PAdES に従った署名を作成することができます。たとえば、以下のコマンドラインは、PAdES パート 3（PAdES Level B-B）に従った基本署名を作成します：

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

以下のコマンドラインは、明示的ポリシー識別子を伴った PAdES パート 3（PAdES Level E-EPES）に従った署名を作成します：

```
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
```

さらに詳しくは 7.7 節「各種 CAdES・PAdES 署名規格」（132 ページ）をごらんください。

2.8 電子署名を視覚化

電子署名を、企業ロゴや手書き署名のスキャン等によって、視覚化することができます。その視覚表現は、PDF 文書として与えられる必要があります。これは署名フォームフィールド内に配置されます。入力文書が署名フィールドをまだ含んでいない場合には、然るべきフィールド座標を与える必要があります。以下のコマンドラインは、視覚化文書 *signing_man.pdf* をフィールド長方形内に配置します：

```
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
```

さらに詳しくは 7.3.1 節「グラフィックかロゴを用いて署名を視覚化」（105 ページ）をごらんください。

2.9 電子署名をクエリ

一般的署名プロパティをクエリ　PLOP DS に内蔵されている pCOS プログラミングインタフェースを用いて、PDF 文書の署名設定をクエリすることができます。pCOS クックブックのトピック ***signatures*** が、署名の種別と内容をクエリする方法を演示しています。pCOS を用いて文書情報をクエリするためのサンプルコードが、すべての PLOP パッケージに含まれている ***dumper*** ミニサンプル内にあります。pCOS コマンドラインツールを使用すると、プログラミングをせずに PDF 文書から情報をクエリできます（1.6 節「pCOS を用いて文書情報をクエリ」（22 ページ）を参照）：

```
pcos *.pdf
```

このプログラム呼び出しは、以下のような出力を生成します：

```
       File name: hellosign.pdf
       File size: 166699
     PDF version: 1.7
       Revisions: 0
       Master pw: false
         User pw: false

      ...
      Tagged PDF: false
      Signatures: 1
        signature field 'Signature1': invisible approval signature, CAdES
  Reader-enabled: false
```

pCOS プログラミングインタフェースで作業している場合には、*signaturefields[ ]* 疑似オブジェクトを使って、PDF 文書内の署名に関する詳細を取得できます（詳しくは pCOS パスリファレンスを参照）。

CMS 署名オブジェクトを抽出

以下のコマンドは、署名済文書から、暗号化の詳細を有する CMS オブジェクトを DER 形式で抽出します：

```
pcos --binary --pcospath signaturefields[0]/V/Contents[0]/V/Contents ↵
        --outfile signature.der input.pdf
```

この抽出された DER 符号化された CMS オブジェクトは、OpenSSL コマンドラインツール等を用いてさらに解析することも可能です：

```
openssl cms -in signature.der -inform DER -cmsout -print
```

3 PLOP・PLOP DS コマンドラインツール

注記 別途のマニュアルで解説しているpCOSコマンドラインツールについても参照してください。

3.1 PLOP・PLOP DS コマンドラインオプション

PLOP・PLOP DS に組み込まれているコマンドラインツールを使うと、一切プログラミングを行う必要なしに、1 個ないし複数の PDF 文書に対して、暗号化・復号・最適化・修復・署名を行うことができます。さらに、これを使って PDF 文書の状態をクエリすることも可能です。PLOP のプログラムを、豊富なコマンドラインオプションで制御することができます。これは 1 個ないし複数の入力 PDF ファイルに対して、次のように呼び出されます（角カッコ内のエントリはオプショナルです）：

```
plop --help
plop [ <一般のオプション群> ] <変換オプション群> --outfile <ファイル名> <ファイル名>
plop [ <一般のオプション群> ] <変換オプション群> --targetdir <パス名> <ファイル名>...
```

PLOP コマンドラインツールは、PLOP ライブラリ上に乗る形で作られています。デフォルトでは PLOP は、破損していることがわかった入力文書については修復を行います。ライブラリのオプションを、8 章「PLOP・PLOP DS ライブラリ API リファレンス」（137 ページ）のオプション一覧に従って、*--inputopt*・*--outputopt*・*--plopopt*・*--signopt*・*--visdocopt* オプションを使って与えることができます。表 3.1 にすべての PLOP コマンドラインオプションを挙げます。

表 3.1 PLOP コマンドラインオプション

オプション	引数	機能
--		オプション群のリストを終了。- キャラクタで始まっているファイル名に対して役立ちます。
@filename[1]		オプション群を内容とする、名前 filename を持つ応答ファイルを指定。文法の詳細は、「応答ファイル」（42 ページ）を参照してください。応答ファイルは、— オプションの前、かつ最初の filename の前でのみ認識され、また、他のオプションの引数を置き換えるために用いることはできません。
--help, -?（またはオプションなし）		利用できるオプションをまとめたヘルプを表示。
--inputopt	< オプションリスト >	PLOP_open_document( ) に対するオプションリスト（表 8.3（144 ページ）参照）
--master, -m	< パスワード >	出力のマスターパスワード。オプションなしはパスワードなしを意味します。
--noreplace, -n		出力ファイルがすでに存在する場合に、上書きされずに例外が発生。デフォルト：出力ファイルがすでに存在していても上書きされます。
--outfile, -o	< ファイル名 >	（ちょうど 1 個の入力文書が必須。--outfile と --targetdir のどちらか 1 つを与える必要があります）出力ファイル名。入力と出力のファイル名は異なる必要があります。

表 3.1 PLOP コマンドラインオプション

オプション	引数	機能
--outputopt	< オプションリスト >	PLOP_create_document( ) に対するオプションリスト（表 8.4（148 ページ）参照）
--password, -p	< パスワード >	入力文書（複数可）のためのユーザーパスワードまたはマスターパスワード。このパスワードがすべての入力文書に対して使われます。必要なパスワードが文書によって異なるときは、それぞれ別個のプログラム呼び出しで処理する必要があります。 --inputopt を用いてデジタル ID が与えられている場合には、このパスワードはその ID に適用されます。
--permissions	< 権限群 >	(--master か --recipient が必須）出力文書に対するアクセス権限リスト。キーワード noprint・nomodify・nocopy・noannots・noassemble・noforms・noaccessible・nohiresprint・plainmetadata を任意の数含みます（表 5.3（68 ページ）参照)。この他に、次のキーワードも使えます（デフォルト：権限制限なし）： ***keep*** 入力文書の権限設定を引き継ぎます。入力文書から引き継いだ権限設定に変更を加えるために、この設定にキーワードを追加して修正条項とすることもできます。例：keep noprint 証明書セキュリティモードでは plainmetadata のみ許容されます。他の権限制限は --recipient オプションリストの permissions オプションで指定できます。
--plopopt	< オプションリスト >	PLOP_set_option( ) に対するオプションリスト（表 8.12（168 ページ）参照)。これを使って、license または licensefile オプションを渡すことが可能です。
--recipient, -r[1]	< オプションリスト >	証明書セキュリティを用いて保護されている文書に対して受信者を追加するための PLOP_add_recipient( ) に対するオプションリスト。このオプションを一回でも与えると証明書セキュリティモードになります。この受信者はすべての入力ファイルに対して用いられます。このオプションを --master/-m・--user/-u とともに使ってはいけません。
--recsize, -R	< ブロックサイズ >	(MVS のみ）出力ファイルのレコードサイズ。デフォルト：0（非ブロック）
--searchpath, -s[1]	< パス >	ファイルの検索されるディレクトリの名前。このパスはマイナスキャラクタ「-」で始めてはいけません（その必要があるときは頭に ./ を付けます)。デフォルト：カレントディレクトリ
--signopt, -S	< オプションリスト >	(PLOP DS でのみ利用可能）文書に電子的に署名するための、PLOP_prepare_signature( ) に対するオプションリスト（表 8.7（155 ページ）参照)。
--targetdir, -t	< ディレクトリ名 >	(--outfile と --targetdir のどちらか 1 つを与える必要があります）出力ディレクトリ名。このディレクトリはすでに存在していなければなりません。
--tempdirname	< ディレクトリ名 >	PLOP の内部処理に必要な一時ファイルが作成されるディレクトリの名前。空にすると、PLOP は一時ファイルをカレントディレクトリに生成します。デフォルト：空
--tempfilename, -T	< ファイル名 >	(MVS のみ）PLOP の内部処理に必要な一時ファイルのフルファイル名。空にすると、PLOP が一意な一時ファイル名を生成します。PLOP が完了した時にこの一時ファイルを削除するのはユーザー側の役割です。デフォルト：空
--user, -u	< パスワード >	(--master を必要とします）出力のユーザーパスワード。オプションなしはパスワードなしを意味します。

表 3.1 PLOP コマンドラインオプション

オプション	引数	機能
--verbose, -v	0, 1, 2, 3	詳細度レベル（デフォルト：1）： *0* 出力なし *1* エラーメッセージのみ *2* エラーメッセージ内にファイル名と API メソッド名を追加 *3* 詳細報告
--visdoc	＜ファイル名＞	（--signopt とともにのみ可）電子署名を視覚化するために用いられるページを含んだ PDF ファイルの名前。
--visdocopt	＜オプションリスト＞	（--visdoc とともにのみ可）署名視覚化文書を開くために用いられる、PLOP_open_document( ) に対するオプションリスト（表 8.3（144 ページ）参照）
--webopt, -w		PDF 出力を Web 配信のために線形化。これは Web 最適化としても知られています。デフォルト：線形化しない

1. このオプションは複数回与えることもできます。

PLOP コマンドラインを組み立て　PLOP コマンドラインを組み立てる際には、以下の規則を守る必要があります：

- 入力ファイルは、*searchpath* として指定されたすべてのディレクトリ内で検索されます。
- オプションによっては短縮形も利用でき、長いオプションと混ぜ書きも可能です。
- 長いオプションは省略もできますが、ただしその省略形は一意でなくてはなりません（例：*--plopopt* のかわりに *--plop*）。
- 1 個のオプションを複数回書くと、最後のものだけが有効とされます。ただし、表 3.1 で複数回与えることもできると注記しているオプションについてはその限りではありません。
- 入力ファイルの暗号化状態によっては、処理のためにはユーザーパスワードかマスターパスワードが必要になります。これは *--password* オプションで与える必要があります。PLOP はこのパスワードが、要請されたアクションに対して十分なものかを調べ（表 5.2 参照）、もしそうでないときは例外を発生させます。

PLOP は、まだどのファイルをも処理しない前に、コマンドライン全体を調べます。コマンドライン上のどの位置のオプションであろうと、その中にオプション文法誤りが見つかったときには、どのファイルも一切処理されません。いずれかのファイルを処理できないときは（必要なパスワードがない等の原因で）、エラーメッセージが出て、PLOP は残りのファイルの処理を継続します。

ファイル名　ブランクキャラクタを含むファイル名は、PLOP のようなコマンドラインツールで用いる際には、ある特殊な取り扱いが必要です。ブランクキャラクタを含むファイル名を処理するためには、ファイル名全体をダブルクォートキャラクタ ″ で囲う必要があるのです。ワイルドカードは標準的な流儀に従って使用できます。たとえば **.pdf* は、所与のディレクトリ内において、ファイル名接尾辞 *.pdf* を持ったすべてのファイルを表します。なお、システムによっては大文字と小文字は区別され、システムによってはされません（すなわち、**.pdf* は **.PDF* と別扱いになる場合があります）。また Windows システムではワイルドカードは、ブランクキャラクタを含むファイル名に対しては働かないことに注意してください。ワイルドカードは、検索パスディレクトリ内では一切評価されず、カレントディレクトリ内で評価されます。

Windows では、すべてのファイル名オプションは Unicode 文字列を受け付けます。たとえばExplorerからファイルをコマンドプロンプトウィンドウへドラッグした場合にはそうなります。

応答ファイル　オプションは、コマンドラインで直接与える方法のほかに、応答ファイルで与える方法もあります。応答ファイルの内容は、コマンドラインの中で、***@filename*** オプションが見つかった位置に挿入されます。

応答ファイルは、オプション群と引数群を記述したシンプルテキストファイルです。以下の文法規則に従う必要があります。

- オプションの複数の値は、空白系文字、すなわちスペース・ラインフィード・リターン・タブのいずれかで区切る必要があります。
- 空白系文字を含む値は、ダブルクォーテーションマーク「"」で囲う必要があります。
- 値の最初と最後のダブルクォーテーションマークは切り捨てられます。
- ダブルクォーテーションマークをリテラルに用いるためには、バックスラッシュでマスクして「\"」とする必要があります。
- バックスラッシュキャラクタをリテラルに用いるためには、もう 1 個のバックスラッシュでマスクして「\\」とする必要があります。

応答ファイルは入れ子にすることもできます。すなわち、応答ファイルの中で ***@filename*** を用いて別のライセンスファイルを参照することも可能です。

応答ファイルは、ファイル名・パスワードオプションに対して、Unicode 文字列を含むことが可能です。応答ファイルは UTF-8・EBCDIC-UTF-8・UTF-16 のいずれかの形式で符号化することができ、対応する BOM で始まっている必要があります。BOM が見つからないときは、応答ファイルの内容は、zSeries では EBCDIC として、それ以外のすべてのシステムでは ISO 8859-1（Latin-1）として解釈されます。

終了コード　PLOP コマンドラインツールは終了コードを返しますので、それを使えば、指示した操作が成功裏に実行されたかどうかを調べることができます：

- 終了コード 0：すべてのコマンドラインオプションと入力ファイルが、成功裏に、かつ完全に処理された。
- 終了コード 1：1 個ないし複数のファイル処理エラーが起きたが、処理は継続された。
- 終了コード 2：コマンドラインオプション内に何らかのエラーが見つかった。処理はその特定の悪いオプションの位置で停止し、どの文書も一切処理されていない。

3.2 PLOP・PLOP DS コマンドラインの作成例

以下の作成例は、PLOP コマンドラインオプションのいくつかの役立つ組み合わせを示しています。すべてのサンプルは 2 つの形式で示してあり、1 番目ではすべてのオプションの長い形式を用いていますが、2 番目では同等の短いオプション形式を用いています。この他にも、以下の節で作成例が得られます：

- 1 章「PLOP の諸機能」（17 ページ）（さまざまな節）
- 5.3 節「コマンドラインでパスワードセキュリティを適用」（70 ページ）
- 6.5 節「コマンドラインで証明書セキュリティを適用」（87 ページ）
- 7.2 節「PLOP DS を用いて署名する」（95 ページ）

ディレクトリ内のすべての PDF 文書を線形化し（これらはどれもパスワードが不要と前提）、できたファイルをターゲットディレクトリ *output* へコピー。詳細度レベル 2 は、すべての入力・出力ファイルについて、その処理時に名前を印字します：

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

カレントディレクトリ内のすべてのファイルを、同一のユーザーパスワード *demo* とマスターパスワード *DEMO* で暗号化し、できたファイルをターゲットディレクトリ *output* に置きます：

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

1 個の受信者証明書に対して文書を暗号化：

```
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
```

ファイル *demo_signer_rsa_2048.p12* の中のデジタル ID を使って、PDF 文書に不可視の署名を作成。デジタル ID に対するパスワードは、ファイル *pw.txt* に入っています：

```
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
```

署名を作成し、手書き署名を内容とする既存 PDF をその署名を視覚化するために使用：

```
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 PLOP・PLOP DS ライブラリの言語バインディング

4.1 C バインディング

PLOP は、C にいくつかの C++ モジュールを加えて記述されています。C バインディングを使用するには、静的または共有ライブラリ（DLL/SO）を使用することができ、中央 PLOP インクルードファイル ***ploplib.h*** を自分のクライアントソースモジュールにインクルードする必要があります。あるいは、***ploplibdl.h*** を用いて PLOP DLL を実行時に動的に読み込むこともできます（詳しくは次項を参照）。

注記 PLOP の C バインディングを使用するアプリケーションは、C++ コンパイラでリンクを行う必要があります。このライブラリは C++ で実装されている部分をいくつか含んでいるからです。C リンカを使用すると、未解決の外部実体が生じる可能性があります。ただし、必要なC++対応ライブラリ群に対してアプリケーションが明示的にリンクされればこの限りではありません。

エラー処理　PLOP API では、ライブラリが発生させる例外に対処する機構を提供しています。これは、C 言語にはネイティブな例外処理がないことを補うためです。***PLOP_TRY()***・***PLOP_CATCH()*** マクロを使用することによって、例外が発生したときにエラー処理とクリーンアップのための専用のコード群が呼び出されるようにクライアントコードを作ることができます。これらのマクロは 2 つのコードセクションを作ります : 例外を発生させる可能性のあるコードを持った try 節と、例外に対処するコードを持った catch 節です。try ブロック内で呼び出された API メソッドのいずれかが例外を発生させたときは、プログラムの実行は catch ブロックの先頭ステートメントへただちに引き継がれます。PLOP クライアントコード内で以下の規則を守る必要があります :

- ***PLOP_TRY()*** と ***PLOP_CATCH()*** は必ず対にする必要があります。
- ***PLOP_new()*** が例外を発生させることは一切ありません。try ブロックは有効な PLOP オブジェクトハンドルでのみ開始できますので、***PLOP_new()*** への呼び出しはあらゆる try ブロックの外で行う必要があります。
- ***PLOP_delete()*** が例外を発生させることは一切ありませんので、try ブロックの外で呼び出しても安全です。catch 節の中で呼び出すこともできます。
- try ブロックと catch ブロックの両方で用いられる変数については特に注意が必要です。コンパイラは 1 個のブロックから別のブロックへの制御の遷移について知りませんので、この場合には不適切なコードが生成される可能性があります（レジスタ変数最適化など）。
  幸い、この種の問題を避けるための簡単な規則があります : try ブロックと catch ブロックの両方で用いられる変数は ***volatile*** 宣言する必要があります。***volatile*** キーワードを使用することで、コンパイラに対して、危険な最適化をこの変数に対して適用しないよう伝達することがでいます。
- try ブロックを去る場合には（return ステートメントなどによって、すなわち対応する ***PLOP_CATCH()*** への呼び出しをバイパスして）、例外機構に知らせるために、return ステートメントの前に ***PLOP_EXIT_TRY()*** を呼び出す必要があります。
- すべての PLOP 言語バインディングの場合と同様、例外が発生したときには文書処理は停止する必要があります。

以下のコードはこれらの規則を、クライアントコード内で PLOP 例外を扱う典型的イディオムとともに演示しています（完全なサンプルが PLOP パッケージ内にあります）：

```
if ((plop = PLOP_new()) == (PLOP *) 0)
{
        printf("PLOPオブジェクトを生成できませんでした。メモリ不足です\n");
        return(2);
}
PLOP_TRY(plop)
{
        /* APIメソッド群を直接または間接に呼び出すステートメント群 */
}
PLOP_CATCH(plop)
{
        printf("エラー %d が %s() で発生しました: %s\n",
            PLOP_get_errnum(plop), PLOP_get_apiname(plop), PLOP_get_errmsg(plop));
}
PLOP_delete(plop);
```

名前文字列に対する Unicode の扱い　C プログラミング言語は、バージョン C11 においてのみ、真の Unicode 文字列に対応しています。このバージョンはまだ広く普及しているとは言いがたいことから、PLOP/PLOP DS は、昔ながらの ***char*** データ型に基づいた Unicode 対応を提供しています。API メソッドの文字列引数のなかには、**名前文字列**として宣言されているものもあります。これらは、***length*** 引数の存在と、文字列の先頭の BOM の存在に従って扱われます。C では、***length*** 引数が 0 でないときは、その文字列は UTF-16 として解釈されます。***length*** 引数が 0 のときは、その文字列は、UTF-8 BOM で始まっていれば UTF-8 として、EBCDIC UTF-8 BOM で始まっていれば EBCDIC UTF-8 として、BOM が見つからなければ ***host*** エンコーディングとして（EBCDIC ベースのプラットフォーム群では ***ebcdic*** として）解釈されます。

オプションリストに対する Unicode の扱い　オプションリスト内の文字列には特に注意が必要です。UTF-16 形式の Unicode 文字列として表現することができず、バイト列としてのみ表現できるからです。この理由から、Unicode オプションに対しては UTF-8 が用いられています。オプションの先頭に BOM を探すことによって、PLOP はそれをどのように解釈するかを決定します。この BOM を用いて文字列の形式が決定されます。より厳密には、文字列オプションの解釈は以下のように働きます：

- オプションが UTF-8 BOM（***\xEF\xBB\xBF***）で始まっていれば、それは UTF-8 として解釈されます。
- オプションが EBCDIC UTF-8 BOM（***\x57\x8B\xAB***）で始まっていれば、それは EBCDIC UTF-8 として解釈されます。
- BOM が見つからないときは、その文字列は ***winansi*** として（EBCDIC ベースのプラットフォームでは ***ebcdic*** として）扱われます。

注記 ***PLOP_convert_to_unicode()*** **ユーティリティメソッドを使うと、UTF-16 文字列から UTF-8 文字列を生成することができます。これは Unicode 値を持つオプションリストを作成するのに役立ちます。**

PLOP を実行時に読み込まれる DLL として使用　多くのクライアントでは PLOP を、静的結合ライブラリとして、またはリンク時に結合される動的ライブラリとして使用しますが、DLL を実行時に読み込んで、すべての API メソッドへのポインタを動的に取得することも可能です。これは特に、必要時にのみ DLL を読み込むのに役立ちます。PLOP では、

この動的使用を実現するための特殊な機構を用意しています。これは以下の規則に従って利用できます：

- *ploplib.h* でなく *ploplibdl.h* をインクルードします。
- *PLOP_new()*・*PLOP_delete()* でなく *PLOP_new_dl()*・*PLOP_delete_dl()* を使用します。
- *PLOP_TRY()*・*PLOP_CATCH()* でなく *PLOP_TRY_DL()*・*PLOP_CATCH_DL()* を使用します。
- 他のすべての PLOP 呼び出しに対して関数ポインタを使用します。
- 追加モジュール *ploplibdl.c* をコンパイルし、できたオブジェクトファイルに対して自分のアプリケーションをリンクします。

この動的読み込み機構は *encryptdl.c* サンプルで演示されています。

4.2 C++ バインディング

ploplib.h C ヘッダファイルに加えて、C++ 用のオブジェクト指向ラッパが PLOP クライアントのために提供されています。これは *plop.hpp* ヘッダファイルを必要としており、このヘッダファイルは *ploplib.h* をインクルードしています。*plop.hpp* はテンプレートベースの実装となっていますので、対応する *plop.cpp* モジュールは不要です。C++ オブジェクトラッパを利用することで、すべての PLOP 関数名に PLOP_ 接頭辞が付いた API メソッドによる関数的アプローチを、よりオブジェクト指向のアプローチへ置き換えることができます。

C++ での文字列処理　PLOP のテンプレートベースのアプローチで、文字列処理に関して以下の使用パターンが可能です：

- C++ 標準ライブラリ型 *std::wstring* の文字列が基本文字列型として用いられます。これは、UTF-16 または UTF-32 で符号化された Unicode キャラクタを持つことができます。これはデフォルト動作であり、カスタムデータ型（次項参照）が *wstring* に対して大きな利点を持たない限り、新しいアプリケーションに対する推奨アプローチです。
- 文字列処理のためのカスタム（ユーザー定義）データ型を、そのカスタムデータ型が *basic_string* クラステンプレートのインスタンス化であり、かつユーザーが与える変換メソッドによって Unicode との相互変換が可能である限り、用いることができます。

デフォルトのインタフェースは、PLOP メソッドとやりとりされるすべての文字列がネイティブ *wstring* であると見なします。*wchar_t* データ型のサイズによって、*wstring* は UTF-16 で（2 バイトキャラクタ群）、または UTF-32 で（4 バイトキャラクタ群）符号化された Unicode 文字列を内容として持つと見なされます。ソースコード内のリテラル文字列は、ワイド文字であることを示すために先頭に *L* を付ける必要があります。リテラル内で Unicode キャラクタは *\u*・*\U* 文法で作成できます。この文法は標準 ISO C++ に含まれているのですが、コンパイラによってはこれに対応していないものがあります。その場合にはリテラル Unicode キャラクタは 16 進キャラクタで作成する必要があります。

注記 EBCDIC ベースのシステム群では、wstring ベースのインタフェースのためのオプションリスト文字列のフォーマッティングは、オプションリスト内に EBCDIC と UTF-16 の wstring が混在することを避けるために、さらなる変換を必要とします。この変換のための簡便なコードと使用法が、追加モジュール utf16num_ebcdic.hpp 内にあります。

C++ でのエラー処理　PLOP API メソッドはエラー発生時に C++ 例外を投げます。これらの例外はクライアントコード内で C++ の *try/catch* 節を用いてキャッチする必要があります。さらなるエラー情報を提供するために、PLOP クラスはパブリックな *PLOP::Exception* クラスを提供しており、このクラスは、詳細なエラーメッセージ、例外番号、例外を発生させた PLOP API メソッドの名前を取得するためのメソッドを公開しています。

PLOP ルーチンが発生させたネイティブな C++ 例外は期待どおりに動作します。以下のコードは、PLOP が発生させた例外をキャッチします：

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

PLOP を実行時に読み込まれる DLL として使用　C 言語バインディングと同様、C++ バインディングでも、PLOP を自分のアプリケーションに実行時に動的に結合させることができます（「PLOP を実行時に読み込まれる DLL として使用」（46 ページ）を参照）。動的読み込みは、*plop.hpp* をインクルードするアプリケーションモジュールをコンパイルする際に下記のようにして有効にすることができます：

```
#define PLOPCPP_DL    1
```

これに加え、追加モジュール *ploplibdl.c* をコンパイルし、できたオブジェクトファイルに対して自分のアプリケーションをリンクする必要があります。動的読み込みの詳細は PLOP オブジェクト内に隠されていますので、それは C++ API に影響を与えません：動的読み込みが有効にしてあってもなくても、すべてのメソッド呼び出しは同じに見えます。

4.3 Java バインディング

PLOP の Java 版をインストール　PLOP/PLOP DS はネイティブな C ライブラリとして実装されており、Java には JNI（Java Native Interface）を通じてアタッチします。当然、Java アプリケーションを開発するには、JNI への対応を含んだ JDK が必要です。PLOP バインディングが動作するためには、PLOP の Java ラッパライブラリと PLOP の Java パッケージが、Java VM から見えている必要があります。

PLOP の Java パッケージ　Java の開発者にとって整合性のあるルックアンドフィールを保つため、PLOP は次のパッケージ名を持った Java パッケージとして構成されています：

```
com.pdflib.plop
```

このパッケージは *plop.jar* ファイルの中にあり、*plop* という 1 個のクラスを含んでいます。PLOP をさまざまな Java 開発環境で使用するにあたっての最新情報が、*readme.txt* ファイル内にあるかもしれません。

このパッケージを自分のアプリケーションに与えるには、自分の *CLASSPATH* 環境変数に *plop.jar* を追加するか、または Java コンパイラ・ランタイムへの呼び出しに *-classpath plop.jar* オプションを追加するか、ないしはそれと同等の手順を Java IDE 内で踏む必要があります。Java VM の設定として、*java.library.path* プロパティにディレクトリの名前を設定すれば、そのディレクトリでネイティブライブラリが検索されるようにすることができます。たとえば

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

このプロパティの値は次のようにして知ることができます：

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

このほかに、以下のようなプラットフォーム独自の手順を踏む必要があります：

- Unix：ライブラリ *libplop_java.so* を、共有ライブラリのためのデフォルトの場所のうちのいずれか 1 つに、または適切に設定されたディレクトリに置く必要があります。
- macOS：ライブラリ *libplop_java.jnilib* を、共有ライブラリのためのデフォルトの場所のうちのいずれか 1 つに、または適切に設定されたディレクトリに置く必要があります。
- Windows：ライブラリ *plop_java.dll* を、Windows のシステムディレクトリに、または PATH 環境変数に示されているディレクトリに置く必要があります。

PLOP サーブレットと Java アプリケーションサーバ　PLOP/PLOP DS は、サーバサイドの Java アプリケーションに、とりわけサーブレットに完全に適合しています。特定のサーブレットエンジンで PLOP を使用する際には、以下の設定上のきまりを守る必要があります：

- サーブレットエンジンがネイティブライブラリを探すディレクトリは、ベンダによって異なります。よくある候補としては、システムディレクトリや、背後の Java VM に特有のディレクトリ、サーブレットエンジンのローカルディレクトリが挙げられます。自分のサーブレットエンジンのベンダから提供されている説明書を見てください。
- サーブレットをロードするのが特別なクラスローダで、制限されていたり、専用のクラスパスを使用していたりすることはよくあります。サーブレットエンジンによっては、特別なエンジンクラスパスを定義して、PLOP パッケージが確実に見つかるようにする必要があります。

PLOP ディストリビューションには、PLOP をサーブレット内で使用している例が入っています。

Java での例外処理　PLOP/PLOP DS のメソッドはすべて、エラー時には *PLOPException* 型の例外を発生させます。PLOP の利用者は、標準的な Java 言語の機能を使ってその例外をキャッチし、それに対処することができます。

```
try {
        plop plop;
        /* ... 各種PLOP命令 ... */

} catch (PLOPException e) {
        System.err.println("暗号化: PLOP例外が発生しました:");
        System.err.println(e.get_apiname() +": " + e.get_errmsg());
} catch (Exception e) {
        System.err.println(e);
} finally {
        /* PLOPオブジェクトを削除 */
        if (plop != null) plop.delete();
}
```

4.4 .NET バインディング

4.4.1 .NET バインディングの種類

PLOP/PLOP DS の .NET 版バインディングには 2 つの種類があります：

- .NET Core バインディング：C# Interop を基盤としています
- クラシック .NET バインディング：C++ Interop を基盤としています

両 .NET バインディングは、具体的実装と対応ターゲット環境について、表 4.1 に示すとおりの違いがあります。この情報に基づいて、自分のアプリケーションに最適のバインディングを選択できます。

表 4.1 クラシック .NET バインディングと .NET Core バインディングの比較

	クラシック .NET バインディング：C++ Interop を基盤としています	.NET Core バインディング：C# Interop を基盤としています
ダウンロードパッケージ	Windows インストーラ	プラットフォームごとに異なる zip または tar.gz パッケージ
パッケージの内容	アセンブリ・ドキュメンテーション・サンプル群	アセンブリ・ドキュメンテーション・サンプル群を持った NuGet パッケージ
実装	アンマネージドコードを用いた C++/CLI アセンブリ PLOP_dotnet.dll	マネージドコードを用いた C# アセンブリ PLOP_dotnet.dll と、アンマネージドコードを用いた補助 DLL PLOP_dotnetcore_native.dll
.NET 統合	暗黙の PInvoke を通じた C++ Interop	明示的 PInvoke を通じた C# Interop
.NET Framework 対応	.NET Framework 4.x	.NET Framework 4.6.1 以上
.NET Core 対応	なし	.NET Standard 2.0
ターゲットオペレーティングシステム	Windows x86・x64	Windows x64・Linux Intel 64・macOS
Windows のレジストリの扱い	インストーラが、レジストリに PLOP を登録し、ライセンスキーを追加します	登録は不要。ライセンスキーのためのレジストリエントリについては手作業で追加する必要があります
クラス名	PLOP_dotnet	PLOP_dotnet

4.4.2 .NET Core バインディング

PLOP の .NET Core 版は、.NET Standard 2.0 に対応しており、ひいては、.NET Core 2.0 以上・.NET Framework 4.6.1 以上・Mono 5.4 以上など多数の環境に対応しています。使用したいターゲットプラットフォームのための .NET Core SDK が必要です。

.NET Core バインディングで用いているバージョン方式は .NET バージョニング規則に準拠しています。NuGet キャッシュや *.csproj* プロジェクトファイルの中で見えているのは .NET Core バージョン番号です。これらのバージョン番号は、PLOP のメジャー・マイナーリリース番号と等しくありません。両バージョニング方式間のマッピングが *compatibility.txt* 内にあります。

この製品は NuGet パッケージとして提供されており、以下のいずれかの手法を用いてローカルにインストールできます：

- ***dotnet*** コマンドラインツール（すべてのプラットフォーム）。この手法については次項で詳しく述べます。
- Visual Studio のパッケージマネージャー UI（Windows・macOS）
- Visual Studio のパッケージマネージャーコンソール（Windows）
- ***nuget*** コマンドラインツール（すべてのプラットフォーム）

提供しているサンプル群のためのプロジェクトファイル群は、ターゲットフレームワーク .NET Core 2.0（ターゲットフレームワークモニカー TFM ＝ ***netcoreapp2.0***）に対して作成されています。別のフレームワークをターゲットしたい場合にはプロジェクトファイル内の TFM を調整するとよいでしょう。例：

```
<PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>netcoreapp3.0</TargetFramework>
</PropertyGroup>
```

PLOP の .NET Core 版を dotnet コマンドラインツールでインストール　***dotnet*** ユーティリティを用いてインストール・構成・ビルドを行う過程を説明します。提供している ***encrypt*** プロジェクトを例に用います：

- 圧縮されている製品パッケージを、好きなディレクトリへ解凍します。
- コマンドシェルで ***encrypt*** プロジェクトディレクトリへ ***cd*** します：

  ```
  cd <インストールディレクトリ>\bind\dotnetcore\C#\encrypt
  ```

- （提供しているサンプル群は、ローカルの NuGet.Config ファイルを用いてこのパッケージを参照しているので、このステップは不要です）上記 NuGet パッケージを、アプリケーションのプロジェクトディレクトリへ複製します：

  ```
  <インストールディレクトリ>/bind/dotnetcore/PLOP_dotnet.X.Y.Z.nupkg
  ```

- （提供しているサンプル群は、PLOP への参照をすでに含んでいるので、このステップは不要です）次のコマンドを、適切なバージョン番号を入れて入力します（このバージョン番号は、上記 ***.nupkg*** ファイルの名前の中にあります）：

  ```
  dotnet add package PLOP_dotnet X.Y.Z
  ```

  このコマンドは ***.csproj*** プロジェクトファイルに PLOP 参照を追加します。また、ローカルの NuGet パッケージキャッシュの中へ PLOP を、まだ存在しない場合にはインストールします。例：

  ```
  ~/.nuget/packages/plop_dotnet/X.Y.Z
  ```

  このキャッシングがありますので、****.nupkg*** を複製する必要があるのは、最初のプロジェクトに対してだけです。その後のプロジェクト群では、このパッケージファイルはこのキャッシュから取られますので不要です。
- これで、***encrypt*** プロジェクトをビルドして走らせて試せます：

  ```
  dotnet build
  dotnet run
  ```

  結果として、アプリケーションディレクトリ内に、暗号化された出力ファイルが現れます。

4.4.3 クラシック .NET バインディング

注記 クラシック .NET バインディングに関する詳しい情報が、PDFlib-in-.NET-HowTo.pdf 文書にあります。この文書は、ディストリビューションパッケージにあるほか、PDFlib Web サイトにもあります。

クラシック .NET バインディングをインストール PLOP を、提供しているインストーラでインストールします。このインストーラは、PLOP アセンブリと追加データファイル群・説明書・サンプルを、マシン上に対話的にインストールします。このインストーラは PLOP の登録も行なって、Visual Studio の「**参照の追加**」ダイアログボックスの .NET タブで簡単に参照できるようにします。

C# プロジェクト内で .NET バインディングを参照 C# プロジェクト内で .NET バインディングを使用するには、Visual C# .NET 内で、PLOP アセンブリへの参照を、次の手順で作成する必要があります：**「プロジェクト」→「参照の追加 ...」→「参照 ...」**を選択し、インストレーションディレクトリから *PLOP_dotnet.dll* を選択。コマンドラインコンパイラでは、次の例のようにして PLOP を参照できます：

```
csc.exe /r:..\..\bin\PLOP_dotnet.dll encrypt.cs
```

4.4.4 .NET バインディングをアプリケーションで使用

この項の内容は、.NET バインディングのどちらの種類にもあてはまります。すぐ使用できるよう構成された完全な作例群が、すべてのパッケージに含まれています。

.NET バインディングが適切に参照された後は、*PLOP_dotnet.PLOP*・*PLOP_dotnet.PLOPException* クラスを使用できます。

エラー処理 .NET バインディングは .NET の例外に対応しており、実行時の問題が発生したときには、詳細なエラーメッセージのついた例外を投げます。このような例外をキャッチして、それに対して適切に対処するのは、クライアント側の役割です。それをしないと、.NET フレームワークがその例外をキャッチして、通常はアプリケーションを中断させます。

例外関連の情報を伝達するために、PLOP ではそれ自身の例外クラス *PLOP_dotnet.PLOPException* を定義しており、メンバ *get_errnum*・*get_errmsg*・*get_apiname* を持たせています。PLOP はインタフェースを実装していますので、クライアントは *Dispose()* メソッドを呼ぶことでクリーンアップが可能です。

クライアントコードは、PLOP が投げた .NET 例外を、通常の *try...catch* 構文で処理できます：

```
try {
        plop = new PLOP();
        ...各種PLOP命令...
} catch (PLOPException e) {
        // PLOPが投げた例外をキャッチした
        Console.WriteLine("encryptサンプル内でPLOP例外が発生しました:");
        Console.WriteLine("[{0}] {1}: {2}\n",
            e.get_errnum(), e.get_apiname(), e.get_errmsg());
} finally {
        if (plop != null) {
            plop.Dispose();
        }
}
```

4.5 Objective-C バインディング

C・C++ 言語バインディングを Objective-C で使用することもできますが、Objective-C 用の純正の言語バインディングも利用できます。以下の種類の PLOP フレームワークがあります：

- ***PLOP***：macOS で使用
- ***PLOP_ios***：iOS で使用

どちらのフレームワークも、C・C++・Objective-C 用の言語バインディングを内容として持っています。

PLOP の Objective-C 版を macOS にインストール　自分のアプリケーションの中で PLOP を使用するには、***PLOP.framework*** か ***PLOP_ios.framework*** をディレクトリ ***/Library/Frameworks*** へ複製する必要があります。他の場所へ PLOP フレームワークをインストールすることも可能ですが、Apple の ***install_name_tool*** を使用する必要があります。それについてここでは説明しません。PLOP メソッド宣言を有する ***PLOP_objc.h*** ヘッダファイルをアプリケーションソースコードへインポートする必要があります：

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

または

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

PLOP/PLOP DS フレームワークをアプリに埋め込むためには、XCode のコード署名は、バージョン番号 ***A*** を有するフレームワークを期待しますが、PDFlib は数字のバージョン番号を使用しています。この問題を回避するには、以下のように適切に名づけたフレームワークフォルダを作成することができます：

```
cd PLOP.framework/Versions
mv 5.3 A
rm Current
ln -s A Current
```

引数の命名規則　PLOP メソッドの呼び出しの際には、引数を以下の規則に従って与える必要があります：

- 1 個目の引数の値は、メソッド名の直後に、コロンキャラクタ 1 個で区切って与えます。
- それより後の各引数については、それぞれ、その引数の名前とその値を（これもコロンキャラクタ 1 個で互いを区切って）与える必要があります。さまざまな引数の名前は、8 章「PLOP・PLOP DS ライブラリ API リファレンス」（137 ページ）と ***PLOP_objc.h*** 内にあります。

たとえば、API 解説における以下の行は：

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

以下の Objective-C メソッドに照応します：

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

ですので、アプリケーションからは以下のような呼び出しを行う必要があります：

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

コード補完のための Xcode Code Sense を PLOP フレームワークで利用できます。

Objective-C におけるエラー処理　Objective-C バインディングは、PLOP エラーをネイティブ Objective-C 例外へ翻訳します。実行時の問題が起きた場合には、PLOP はクラス *PLOPException* のネイティブ Objective-C 例外を発生させます。これらの例外は、通常の *try/catch* 機構を用いて処理できます：

```
@try {
        ...各種PLOP命令...
}
@catch (PLOPException *ex) {
        NSString * errorMessage =
            [NSString stringWithFormat:@"PLOPエラー %d が '%@'で発生しました: %@",
            [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];
}
```

この *get_errmsg* 方式以外にも、例外オブジェクトの *reason* フィールドを用いてエラーメッセージを取得することもできます。

4.6 Perl バインディング

Perl 用 PLOP ラッパは、1 個の C ラッパと 2 個の Perl パッケージモジュールから成ります。このモジュールの 1 個は各 PLOP API メソッドと同等のものを Perl で提供するもので、もう 1 個は PLOP オブジェクトのためのものです。C モジュールは、Perl インタプリタが実行時に読み込む共有ライブラリを、パッケージファイルからいくらかの助けを借りてビルドするために用いられます。Perl スクリプトは共有ライブラリモジュールを、*use* ステートメントを通じて参照します。

PLOP の Perl 版をインストール　Perl 拡張機構は共有ライブラリを実行時に、DynaLoader モジュールを通じて読み込みます。Perl 実行形式が、共有ライブラリに対応した形でコンパイルされている必要があります（多くの Perl 設定ではそのようになっています）。

PLOP バインディングが動作するためには、Perl インタプリタは PLOP Perl ラッパとモジュール *plop_pl.pm*・*PDFlib/PLOP.pm* を利用可能である必要があります。以下に説明するプラットフォーム固有の方式のほかに、Perl の *@INC* モジュール検索パスに、*-I* コマンドラインオプションを用いてディレクトリを追加することも可能です：

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

Unix　Perl は、*plop_pl.so*（macOS では *plop_pl.bundle*）・*plop_pl.pm*・*PDFlib/PLOP.pm* を、カレントディレクトリ内で、あるいは下記 Perl コマンドで印字されるディレクトリ内で検索します：

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

Perl はサブディレクトリ *auto/plop_pl* も検索します。上記コマンドの典型的出力は下記のようになります：

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

Windows　DLL *plop_pl.dll* とモジュール *plop_pl.pm*・*PDFlib/PLOP.pm* が、カレントディレクトリ内で、あるいは下記 Perl コマンドで印字されるディレクトリ内で検索されます：

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

上記コマンドの典型的出力は下記のようになります：

```
C:\Program Files\Perl5.16\site\lib
```

Perl での例外処理　PLOP の例外が発生すると、Perl の例外が発生します。これは以下のように、*eval* シーケンスを用いて捕捉・対処できます：

```
eval {
        ...各種PLOP命令...
};
die "例外をキャッチしました: $@" if $@;
```

4.7 PHP バインディング

注記 PLOP を PHP で使う際のさまざまな種別やオプションに関する詳しい情報は、ディストリビューションパッケージ内に含まれている、PDFlib の Web サイトにもある *PDFlib-in-.NET-HowTo.pdf* 文書に掲載しています。これは主に PDFlib を PHP で使う際のことを述べていますが、その説明は PLOP を PHP で使う場合についても同様に当てはまります。

PLOP の PHP 版をインストール　PLOP/PLOP DS は、PHP へ動的にアタッチできる C ライブラリとして実装されています。PLOP は PHP のいくつかのバージョンに対応しています。アンパックした PLOP アーカイブの中から、自分が使う PHP のバージョンに合わせて、適切な PLOP ライブラリを選ぶ必要があります。

PHP が外部の PLOP ライブラリを認識するよう PHP を構成する必要があります。2 つの選択肢があります：

- *php.ini* に以下のいずれかの行を追加：

```
extension=plop_php.so        ; Unix・macOS用
extension=plop_php.dll       ; Windows用
```

PHP はこのライブラリを、Unix の場合は *php.ini* 内の *extension_dir* 変数で指定されているディレクトリで検索し、Windows の場合はそこのほかに標準のシステムディレクトリ群でも検索します。どのバージョンの PLOP の PHP バインディングをインストールしてあるかは、下記の 1 行の PHP スクリプトで調べることができます：

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

これは、自分の現在の PHP 設定に関する長い情報ページを表示します。このページで、*plop* と題されたセクションを調べます。もしこのセクションに下記

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

（この後に PLOP のバージョン番号）があれば、PLOP の PHP 版を正しくインストールできています。

- 自分のスクリプトの先頭に、以下のいずれかの行を書いて、PLOP を動作時にロードする：

```
dl("plop_php.so");           # Unix・macOS用
dl("plop_php.dll");          # Windows用
```

PHP でのファイル名処理　PDF や画像・フォント等のディスクファイルに対する無修飾のファイル名（パス要素のない）と相対ファイル名は、PHP の Unix 版と Windows 版とでは、扱われ方が異なります：

- Unix 諸システムの PHP の場合、パス要素を持たないファイルは、スクリプトが置かれているディレクトリで検索されます。
- Windows の PHP の場合、パス要素を持たないファイルは、PHP DLL が置かれているディレクトリでのみ検索されます。

PHP での例外処理　PHP では、構造化された例外処理に対応しているので、PLOP の例外は PHP の例外として伝達されます。標準的な *try/catch* 技法を使って PLOP 例外を取り扱えます：

```
try {

...各種PLOP命令...

} catch (PLOPException $e) {
    print "PLOP例外が発生しました:\n";
    print "[" . $e->get_errnum() . "] " . $e->get_apiname() . ": "
                $e->get_errmsg() . "\n";
}
catch (Exception $e) {
    print $e;
}
```

Eclipse と Zend Studio で開発　PHP Development Tools（PDT）は、Eclipse と Zend Studio を用いた PHP 開発に対応しています。PDT は、以下に概略示す操作によって、文脈依存ヘルプに対応するよう構成することもできます。

PLOP がすべての PHP プロジェクトから認識されるよう PLOP を Eclipse 設定に追加します：

- **「ウィンドウ」→「設定」→「*PHP*」→「*PHP* ライブラリー」→「新規 ...」**を選択します。すると、ウィザードが起動します。
- **「ユーザー・ライブラリ名」**に**「*PLOP*」**と入力し、**「外部フォルダーの追加 ...」**をクリックし、フォルダ ***bind\php\Eclipse PDT*** を選択します。

既存または新規の PHP プロジェクトにおいて、PLOP ライブラリへの参照を以下のように追加できます：

- PHP エクスプローラー内でその PHP プロジェクトを右クリックし、**「インクルード・パス」→「インクルード・パスの構成 ...」**を選択します。
- **「ライブラリー」**タブへ移動し、**「ライブラリーの追加」**をクリックして、**「ユーザー・ライブラリー」→「*PLOP*」**を選択します。

これらの操作の後、PLOP メソッドの一覧を、PHP エクスプローラービュー内の ***PHP Include Path/PLOP/PLOP*** ノード下で閲覧することができます。新規の PHP コードを書く時、Eclipse は、すべての PLOP メソッドについて、コード補完と文脈依存ヘルプで支援します。

4.8 Python バインディング

PLOP の Python 版をインストール　Python の拡張機構は、実行時に共有ライブラリを読み込むことによって動作します。PLOP バインディングが動作するためには、Python インタプリタが PLOP Python ラッパを利用可能である必要があります。このラッパは、PYTHONPATH 環境変数内に挙げられているディレクトリ群の中で検索されます。Python ラッパの名前はプラットフォームによって異なります：

- Unix・macOS：*plop_py.so*
- Windows：*plop_py.pyd*

Python のエラー処理　Python バインディングは、PLOP エラーをネイティブな Python 例外へ翻訳する特殊なエラーハンドラをインストールします。この Python 例外は、通常の try/except 技法で扱えます：

```
try:
        plop = PLOP()
        ...各種PLOP命令...
except PLOPException as ex:
    print("PDFlib PLOP例外が発生しました:")
    print("[%d] %s: %s" % (ex.errnum, ex.apiname, ex.errmsg))

except Exception as ex:
    print(ex)

finally:
    if plop:
        plop.delete()
```

4.9 Ruby バインディング

PLOP の Ruby 版をインストール　Ruby の拡張機構は、実行時に共有ライブラリを読み込むことによって動作します。PLOP バインディングが動作するためには、Ruby インタプリタが Ruby 用 PLOP 拡張ライブラリへアクセスできるようになっている必要があります。このライブラリ（Windows・Unix では ***PLOP.so***。macOS では ***PLOP.bundle***）は通常、ローカルの Ruby インストレーションディレクトリの ***site_ruby*** ブランチ内に、すなわち以下のような名前のディレクトリの中にインストールされます：

```
/usr/local/lib/ruby/site_ruby/<バージョン>/
```

ただし、Ruby は他のディレクトリ群へも拡張を探しに行きます。このディレクトリのリストを取得するには以下のルビー呼び出しを使用できます：

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

このリストは通常、カレントディレクトリを含んでいますので、試験目的のためには、単に PLOP 拡張ライブラリとスクリプト群を同一ディレクトリ内に置けば足ります。

Ruby におけるエラー処理　Ruby バインディングは、PLOP 例外をネイティブ Ruby 例外へ翻訳するエラーハンドラをインストールします。この Ruby 例外は通常の ***rescue*** 技法で扱えます：

```
begin
        ...各種PLOP命令...
rescue PLOPException => pe
  print "PLOP例外が暗号化サンプル内で発生しました:\n"
  print "[" + pe.get_errnum.to_s + "] " + pe.get_apiname + ": " + pe.get_errmsg + "\n"
end
```

5 パスワードセキュリティ

5.1 PDF におけるパスワードセキュリティ

PDF のパスワードセキュリティは、以下の保護機能を実現します：

- ユーザーパスワード（開くパスワードとも呼ばれる）を与えないとファイルを開いて閲覧できないようにする。
- マスターパスワード（所有者パスワードまたは権限パスワードとも呼ばれる）を与えないと、セキュリティ設定、すなわち諸権限・ユーザーパスワード・マスターパスワードを一切変更できないようにする。ユーザーパスワードとマスターパスワードを持つファイルは、そのどちらかのパスワードを与えれば開いて閲覧できます。
- 権限設定は、その PDF 文書に対する特定の動作（印刷やテキスト抽出等）を制限する。
- 添付パスワードを指定すると、文書自体の内容本体は暗号化せず、ファイル添付のみを暗号化することができます。

これらの保護機能のうち 1 つでも用いている PDF 文書は暗号化されます。文書のセキュリティ設定を Acrobat で表示または変更するには、それぞれ**「ファイル」→「プロパティ ...」→「セキュリティ」→「詳細を表示 ...」**か**「設定を変更 ...」**をクリックします。図 5.1 に Acrobat のセキュリティ設定ダイアログを示します。

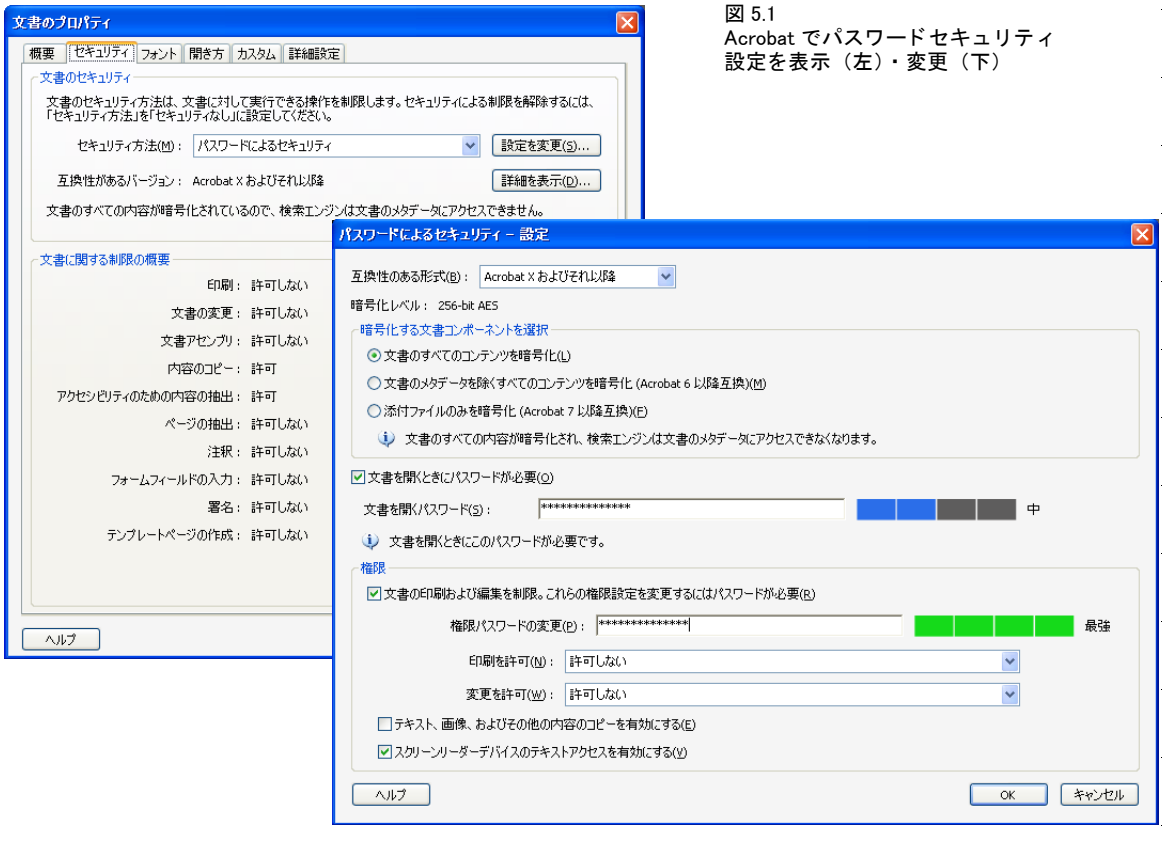


図 5.1
Acrobat でパスワードセキュリティ設定を表示（左）・変更（下）

暗号化アルゴリズムと鍵長　PDF の暗号化は、以下の暗号化アルゴリズムを利用しています：

- RC4。対称ストリーム暗号です（すなわち、同じアルゴリズムを用いて暗号化と復号ができます）。RC4 はもはや充分なセキュリティを実現しません。
- AES（高度暗号化標準）。規格 FIPS-197 で仕様化されています。AES はさまざまな応用で利用されている最新のブロック暗号です。

実際の暗号鍵は扱いにくいバイナリ列なので、それはもっとユーザーフレンドリーなプレーンキャラクタから成るパスワードから導出されます。PDF と Acrobat の発展の過程のなかで、PDF 暗号方式は改良を重ねられ、より強力なアルゴリズム、より長い暗号鍵、より洗練されたパスワードを用いるようになってきています。表 5.1 に、すべての PDF バージョンについて、暗号鍵とパスワードの特徴を示します。

表 5.1 PDF の各バージョンにおける暗号アルゴリズム・鍵長・パスワード長

PDF・Acrobat バージョン、pCOS アルゴリズム番号	暗号アルゴリズムと鍵長	最大パスワード長とパスワードエンコーディング
PDF 1.1 ～ 1.3（Acrobat 2 ～ 4）、pCOS アルゴリズム 1	RC4 40 ビット（脆弱。PDF 2.0 では非推奨）	32 キャラクタ（Latin-1）
PDF 1.4（Acrobat 5）、pCOS アルゴリズム 2	RC4 128 ビット（脆弱。PDF 2.0 では非推奨）	32 キャラクタ（Latin-1）
PDF 1.5（Acrobat 6）、pCOS アルゴリズム 3	PDF 1.4 と同じ RC4 128 ビット、ただし暗号方式の異なる応用（脆弱。PDF 2.0 では非推奨）	32 キャラクタ（Latin-1）
PDF 1.6（Acrobat 7）・PDF 1.7 ＝ ISO 32000-1（Acrobat 8）、pCOS アルゴリズム 4	AES-128（PDF 2.0 では非推奨）	32 キャラクタ（Latin-1）
PDF 1.7ext3（Acrobat 9）、pCOS アルゴリズム 9	AES-256 で、パスワードの扱いに欠陥があるもの（脆弱。PDF 2.0 では非推奨）	127 UTF-8 バイト（Unicode）
PDF 1.7ext8（Acrobat X/XI/DC）・PDF 2.0 ＝ ISO 32000-2、pCOS アルゴリズム 11	AES-256 で、パスワードの扱いが改良されたもの	127 UTF-8 バイト（Unicode）

パスワード　PDF の暗号化は内部的に、PDF バージョンによって 40・120・250 ビットのいずれかの暗号鍵で動作します。ユーザーが与えたパスワードからバイナリ暗号鍵が導出されます。パスワードには長さとエンコーディングの制約があります：

- PDF 1.7（ISO 32000-1）までは、パスワードは最大長 32 キャラクタに限られ、Latin-1 エンコーディング内のキャラクタのみを含むことができます。
- PDF 1.7ext3 では Unicode キャラクタを導入し、最大長を、パスワードの UTF-8 表現で 127 バイトに増やしました。UTF-8 ではキャラクタを可変長 1 ～ 4 バイトに符号化しますので、パスワード内に許される Unicode キャラクタの数は、非 ASCII キャラクタを含む場合には 127 より少なくなります。たとえば、日本語キャラクタは UTF-8 表現では通常 3 バイトを必要としますので、パスワード内で最大 42 個の日本語キャラクタまでが使えることになります。

あいまいさを避けるために、Unicode パスワードは SASLprep という処理（RFC 3454 の Stringprep に基づき RFC 4013 で仕様化されています）によって正規化されます。この処理では、非テキストキャラクタを除去し、ある種のキャラクタクラスを正規化します（たとえば非 ASCII 空白キャラクタは ASCII 空白キャラクタ U+0020 へマップされます）。パス

ワードは Unicode 正規形 NFKC へ正規化され、パスワード内に右書きキャラクタと左書きキャラクタが混在していた場合に起こりうるあいまいさを回避するために特殊な双方向処理が施されます。

PDF 暗号化の強度は、暗号鍵の長さによってのみ決まるのではなく、パスワードの長さと質によっても左右されます。名前や単語そのままなどをパスワードに使うべきではないということは広く知られています。容易に推測できたり、いわゆる辞書アタックによってシステマティックにあたられるからです。さまざまな調査によれば、かなりの数のパスワードは配偶者やペットの名前、ユーザーの誕生日、子供のニックネームなどを用いており、そのため容易に推測可能になっています。

権限制限　PDF では、文書の操作に関するさまざまな制限を符号化することができ、これらは個別に承認または拒否することができます（図 5.1 参照）：

- **印刷を許可**：印刷が許可されていないときは、Acrobat の印刷ボタンは無効になります。Acrobat は、**「低解像度（*150 dpi*）」**と**「高解像度」**の印刷の区別に対応しています。低解像度印刷では、個人的利用にしか適さず、高品位な複製と再 PDF 化を防止する、ページのラスタ画像が生成されます。画像ベースの印刷では出力品質が低くなるだけでなく、印刷処理がかなり遅くなることにも留意してください。
- **変更を許可**：そのリストが、さまざまな文書変更操作に対する制御を実現します：

  ページの挿入、削除、回転
  フォームフィールドの入力と既存の署名フィールドに署名
  注釈の作成、フォームフィールドの入力と既存の署名フィールドの署名
  ページの抽出を除くすべての操作

- 内容コピー操作は、**「テキスト、画像、およびその他の内容のコピーを有効にする」**から制御されます。これは、アクセシビリティのために**「スクリーンリーダーデバイスのテキストアクセスを有効にする」**を用いて有効にすることも可能ですが、この設定は PDF 2.0 では、PDF リーダは常にアクセシビリティに対応するべきであるため、非推奨と見なされます。

「印刷を許可：許可しない」といったアクセス制限を文書に設定すると、Acrobat のそれに照応するメニュー項目が無効になります。しかし、これはサードパーティの PDF ビューアなどのソフトウェアでもそうなるとは限りません。文書内のアクセス権限が実際に効力を持つかどうかは、PDF ツールの開発者にかかっているのです。実際、いくつかの PDF ツールは権限設定を全然無視することで知られています：商用の PDF クラッキングツールを使えば、いかなるアクセス制限も無効化することができます。これは暗号化のクラッキングとは関係ありません：パスワードのない PDF ファイルを、画面では見られても印刷はできないようにすることは、単に不可能なのです。このことは ISO 32000-1 に下記のように記されています：

「ひとたび文書が成功裡に開かれ復号されれば、準拠リーダは技術的にその文書の内容全体にアクセス可能となる。暗号化辞書内で指定されている文書権限設定群を強制できる性質のものは *PDF* 暗号化の中に何もない。」

暗号化された文書構成要素　デフォルトでは、PDF 暗号化は常に 1 個の文書のすべての構成要素をカバーします。しかし、場合によっては、文書内のいくつかの構成要素は暗号化せず、それ以外だけを暗号化したいときもあります：

- PDF 1.5（Acrobat 6）では、プレーンテキストメタデータという機能が導入されました。この機能を使うと、暗号化された文書に、暗号化されていないメタデータを入れ込むことができます。これによって、検索エンジンが文書のメタデータを、暗号化された文書からでも取り出せるようにすることができます。

- PDF 1.6（Acrobat 7）からは、保護されていない文書の中のファイル添付であっても、暗号化することが可能です。これによって、暗号化されていない文書を、秘密の添付のためのコンテナとして利用することができます。

セキュリティ上の推奨事項　以下のことは、生成される暗号が脆弱でクラックされる可能性がありますので、避けるべきです：

- 1 ～ 6 キャラクタから成るパスワードは避けるべきです。可能なすべてのパスワードを試す攻撃（パスワードに対するブルートフォースアタック）に対して脆弱だからです。
- パスワードは単なる単語に似ていてはいけません。可能な単語をすべて試す攻撃（辞書アタック）に対して脆弱だからです。パスワードには非アルファベットキャラクタを含ませるべきです。自分の配偶者やペットの名前、誕生日、その他簡単に推測できる項目を使ってはいけません。
- 脆弱な RC4 アルゴリズムと、PDF 1.7ext3（Acrobat 9）に従った AES-256 は避けるべきです。パスワード確認アルゴリズムに脆弱性を含んでいることから、パスワードに対するブルートフォースアタックが容易なためです。このため、Acrobat XI/DC と PLOP では、新しい文書を保護するために Acrobat 9 の暗号化は決して使用しません（既存の文書を復号するためにのみ使用します）。

まとめると、PDF 1.7ext8/PDF 2.0 に従った AES-256 を使用するべきです。パスワードは 6 キャラクタよりも長くするべきであり、非アルファベットキャラクタを含ませるべきです。

Web 上の PDF を保護　PDF が Web で提供される場合には、ユーザーは必ずその文書のローカルコピーを自分のブラウザで作ることができます。PDF 文書がユーザーにローカルコピーをとられないようにする方法はありません。

5.2 PLOP を用いて PDF 文書をパスワード保護

PLOP は、標準のさまざまなセキュリティ機能を、PDF ファイルに適用したり、PDF ファイルから除去したりします。PLOP では、ユーザーパスワードとマスターパスワードを適用することができ、また、アクセス権限を設定して、Acrobat で文書を印刷できないようにしたり、テキストを抽出できないようにしたり、文書を変更できないようにしたりすることができます。文書を復号するには、適切なマスターパスワードが必要です。

パスワードセキュリティのための暗号アルゴリズムと鍵長 PLOP は常に、AES-128（pCOS アルゴリズム 4）か、またはセキュア版 AES-256（pCOS アルゴリズム 11）を適用します。PLOP は決して、脆弱な RC4 暗号を、あるいは、パスワード処理アルゴリズム内に脆弱性を持つ PDF 1.7ext3/Acrobat 9 に従った脆弱版 AES-256（pCOS アルゴリズム 9）を適用しません。暗号アルゴリズムを選択するには、*PLOP_create_document()* の *encryption* オプションを用います。

- *encryption=algo4* の場合：PDF バージョンは、必要であれば PDF 1.6 へ上げられ、かつ、Acrobat 7/8（pCOS アルゴリズム 4）に従った AES-128 暗号が適用されます。パスワードは Latin-1 キャラクタのみを内容とすることができ、32 キャラクタへ切り詰められます。
- *encryption=algo11* の場合（これがデフォルトです）：PDF バージョンは、必要であれば PDF 1.7ext8 へ上げられ、かつ、Acrobat X/XI/DC（pCOS アルゴリズム 11）に従った AES-256 暗号が適用されます。パスワードは Unicode キャラクタを内容とすることができ、127 UTF-8 バイトへ切り詰められます。

さまざまな PLOP の操作に必要なパスワード PDF 文書の権限設定によって反映されている作成者の意図に厳密に従うためには、パスワードセキュリティを用いて保護されている文書に対する操作をすべて許せるとは限りません。PLOP は以下のルールに従って動作します：

- 暗号化のステータスを pCOS 擬似オブジェクト *encrypt/algorithm* 等を用いてクエリすることは、パスワードに一切よらず、常に可能です。
- 文書のプロパティを pCOS インタフェースを用いてクエリできるかどうかは、pCOS モードによって決まります。たとえば、XMP 文書メタデータ・文書情報フィールド・しおり・注釈内容は、その文書がユーザーパスワードを必要としていない場合には（あるいはユーザーパスワードのみが与えられている場合）、マスターパスワードなしで取得できます。pCOS パスリファレンスで詳しく述べています。
- 以下の操作にはマスターパスワードが必要です：ユーザーパスワード・マスターパスワード・権限設定の変更・除去、暗号化文書に対する線形化・最適化・修復・署名操作。

表 5.2 に、すべての操作について何が必要かをまとめました。

表 5.2 暗号化された文書に対するさまざまな操作のために必要なパスワード

知っているパスワード	暗号化の状態（pCOS 擬似オブジェクト「encrypt」）をクエリ	pCOS を用いて文書情報・XMP メタデータ・しおり・注釈内容をクエリ	パスワード・権限を変更、線形化・最適化・修復・署名
なし	可	ユーザーパスワードが設定されていない場合にのみ可	不可
ユーザー	可	可	不可
マスター	可	可	可

PLOP を用いてパスワードを設定　PLOP ライブラリ API と PLOP コマンドラインオプションでは、元の PDF 文書を**入力**文書と呼び、暗号化または復号された生成物を**出力**文書と呼ぶことにします（どちらも同じファイル名の場合もありますが）。入力文書が保護されている場合、PLOP は表 5.2 に従って、行いたい操作によってユーザーパスワードかマスターパスワードのいずれかを必要とします。入力文書を成功裏に開くことができたならば（保護されていない文書だった場合、または正しいパスワードを与えたことによって）、出力文書にはユーザーパスワード・マスターパスワード・権限設定を任意の組み合わせで適用できます。ただし PLOP は、クライアントが出力文書のために与えるパスワードについて、以下のように作用します：

- ユーザーパスワードか権限設定が与えられているのに、マスターパスワードが与えられていない場合は、通常の利用者がセキュリティ設定を簡単に変えることができ、したがって保護を破れてしまいます。ですので PLOP はこの状況をエラーと見なします。
- ユーザーパスワードとマスターパスワードが同一の場合、ユーザーとファイルの所有者との区別はもはや不可能となり、したがってやはり有効な保護は破れてしまいます。PLOP はこの状況をエラーと見なします。
- AES-256 では Unicode パスワードが許されます。これよりも古い暗号化アルゴリズムでは、Latin-1 文字集合に限られたパスワードを必要とします。古い暗号化アルゴリズムの場合に、与えられたパスワードが Latin-1 文字集合外のキャラクタを含んでいると、例外が発生します。
- パスワードは、AES-256 では 127 UTF-8 バイトまでに、古い暗号化アルゴリズムでは 32 キャラクタまでに切り落とされます。

PLOP で権限を設定　PLOP は、表 5.3 に示す任意の権限設定を、クエリ・設定・削除することができます。特記なき限り、すべての動作はデフォルトでは許されます。アクセス制限を指定すると、Acrobat のそれに対応する機能が無効になります。アクセス制限は、ユーザーパスワードを設定しなくても適用できますが、マスターパスワードは必要です。表 5.3 に、使える権限制限キーワードを列挙します。

表 5.3 PLOP_create_document( )・PLOP_add_recipient( ) の permissions オプションに対する権限制限キーワード

キーワード	説明
文書を印刷	
nohiresprint	高解像度印刷を防止。noprint が指定されていなければ、印刷は、ページの低解像度版を印刷する「画像として印刷」機能に制限されます。
noprint	ファイルの印刷を防止。
文書を変更	
nomodify	フォームフィールドの追加などあらゆる変更を防止。
noannots	注釈の追加・変更とフォームフィールドへの記入を防止。nomodify と noannots が指定されていなければ、フォームフィールド（署名フィールドを含め）の作成・変更は許されます。
noforms	（noannots によって暗黙に指定されます）フォームフィールド記入と署名を防止。noannots が指定されていなくても同様です。
noassemble	（nomodify によって暗黙に指定されます）ページの挿入・削除・回転としおり・サムネールの作成を防止。nomodify が指定されていなくても同様です。
文書から内容をコピー	

表 5.3 PLOP_create_document( )・PLOP_add_recipient( ) の permissions オプションに対する権限制限キーワード

キーワード	説明
nocopy	テキスト・グラフィックのコピー・抽出を防止。
noaccessible	（PDF 2.0 では非推奨）テキスト・グラフィックのアクセシビリティ目的のための抽出を防止。
other	
plainmetadata	（PLOP_create_document( ) のみ）暗号化された文書に対しても文書のメタデータを暗号化しないままにします。
nomaster	（PLOP_add_recipient( ) のみ）指定された権限制限キーワード群に従って印刷・編集・内容抽出を制限し、かつ、文書のセキュリティ設定の変更を防止。このキーワードが与えられていない場合には、受信者はその文書に対して完全な権限を持ちます。すなわちその場合、他のすべての権限制限キーワード（plainmetadata 以外の）は無視されます。

なお、Acrobat は文書の変更に関する 4 種の権限制限すべてに対して完全な制御を実現しているわけではなく、これらの制限のいくつかをグループにまとめています。表 5.4 に、Acrobat の諸設定（図 5.1 内の「変更を許可」リストの値群）と、照応する PLOP の権限制限キーワード群との対照を示します。

表 5.4 Acrobat の権限記述と、PLOP_create_document( )・PLOP_add_recipient( ) の permissions オプションに対する照応するキーワードの組み合わせ

Acrobat の「変更を許可」	照応する権限キーワード
許可しない	nomodify noannots noforms noassemble
ページの挿入、削除、回転	nomodify noannots noforms
フォームフィールドの入力と既存の署名フィールドに署名	nomodify noannots noassemble
注釈の作成、フォームフィールドの入力と既存の署名フィールドに署名	nomodify noassemble
ページの抽出を除くすべての操作	noassemble

5.3 コマンドラインでパスワードセキュリティを適用

文書を暗号化するには、*PLOP_create_document()* で *userpassword* オプションか *masterpassword* オプションを（両方でも可）指定します。ただし、ユーザーパスワードは必ずマスターパスワードを必要としますが、逆は真ではありません。PLOP ライブラリを用いて PDF 文書を保護し、セキュリティを除去する完全なサンプルコードを、すべての PLOP パッケージに入っている *encrypt・decrypt* プログラミングサンプルで見ることができます。PLOPコマンドラインツールでこれと等価なオプションは*--user*と*--master*です。

権限設定は、*PLOP_create_document()* で *permissions* オプションを用いて指定できます。コマンドラインツールでこれと等価なオプションは*--permissions* です。

注記 Windows では、コマンドライン上のパスワードは、Latin-1 文字集合外の Unicode キャラクタを含むことも可能です。

さまざまな暗号化の例 以下のさまざまなサンプルコマンドライン呼び出しは、長いコマンドラインオプションで示しています。短縮形のコマンドラインオプションについては 3.1 節「PLOP・PLOP DS コマンドラインオプション」（39 ページ）を参照してください。

ファイルをユーザーパスワード *demo* とマスターパスワード *DEMO* で暗号化：

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

カレントディレクトリ内のすべてのファイルを、同一のユーザーパスワード *demo* とマスターパスワード *DEMO* で暗号化し、できたファイルをターゲットディレクトリ *output* へ置く：

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

スペースキャラクタを含むパスワードは、次の例のように中カッコで（オプションリスト文法に従うために）、さらにストレートな引用符で（シェル文法に従うために）くくる必要があります。文書をマスターパスワード *two words* で暗号化：

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

文書を暗号化して署名（署名オプションについては 7.2.2 節「内蔵エンジンを用いて署名する」（96 ページ）を参照）：

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

権限設定 マスターパスワード *DEMO* と、権限設定 *noprint・nocopy・noannots* を、ディレクトリ内のすべてのファイルに適用して、できたファイルをターゲットディレクトリ *output* に置く。入力文書で使われている暗号化が何であるかにかかわらず、AES 暗号が適用されます。詳細度レベル 2 では、すべての入力・出力ファイルの名前が、処理されるにつれて印字されます：

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

すべての権限設定をファイルから除去し、その結果を別の出力ファイルへ、同じマスターパスワードでコピー。これには入力文書に対するマスターパスワードが必要です：

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

文書を再暗号化し（たとえば、脆弱な暗号化を強力な AES 暗号へ換えたり、脆弱なパスワードをもっと良いものへ換えたり）、権限設定は入力文書のものを複製。結果を別の出力ファイルへコピー。これには入力文書に対するマスターパスワードが必要です：

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

復号の例　1 個のファイルをマスターパスワード *DEMO* で復号。入力文書にアクセス制限がかけられていたとしても、それらはすべて除去されます（なぜなら出力は暗号化されていないので）：

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

より強力な暗号化方式で再暗号化　PLOP を利用すると、短い鍵や脆弱なパスワードで暗号化されている文書に、もっと強力な暗号化を施すこともできます。旧パスワードと新パスワードを与える必要があります。PLOP はデフォルトで、強力な AES 暗号を使用します。次の例では、入力文書はマスターパスワード *old* で暗号化されているときに、出力をマスターパスワード DEMO で AES 暗号化する場合を想定しています。新しいパスワードは古いパスワードと同じでもかまいません。もちろん、実際にはこの例のような短いパスワードではなく、強力なパスワードだけを用いるべきです（「セキュリティ上の推奨事項」（66 ページ）参照）：

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

6 証明書セキュリティ

6.1 Acrobat における証明書セキュリティ

証明書セキュリティの利点　パスワードセキュリティを用いて保護されている PDF 文書は、ユーザーパスワードかマスターパスワードが知られたら開かれます。難点は、パスワードの配送が、秘密の経路を要することから、困難な場合があることです。また、正当な文書受信者がうっかり、あるいは意図的に、他者へパスワードを漏らしてしまうこともありえます。

証明書セキュリティは、パスワードセキュリティのかわりとなる手段を提供します。これは公開鍵暗号と証明書に基づいています。文書は、特定の受信者たちのために暗号化されます。各受信者は、その者の証明書によって識別されます。証明書は、その公開鍵のみを含んでおり、秘密情報を一切含んでいませんので、保護を全く必要とせず自由に配布できます。保護されている文書を開くには、受信者は、暗号化のために使用された証明書に照応する秘密鍵を有するデジタル ID を必要とします。

証明書セキュリティは、パスワードセキュリティに比べて以下の利点を提供します：

- パスワードを受信者へ配布する必要が一切ない。
- 各受信者に対して、ないしは受信者のグループに対して、個別の権限制限を指定できる。権限は、使用権が互いに異なるユーザーたちへ文書を配布する際に役立ちます。
- 受信者が、権限のない第三者へ文書のパスワードを渡せない。受信者が自分のデジタルソフトウェア ID を複製して渡してしまう可能性はありますが、その ID からその受

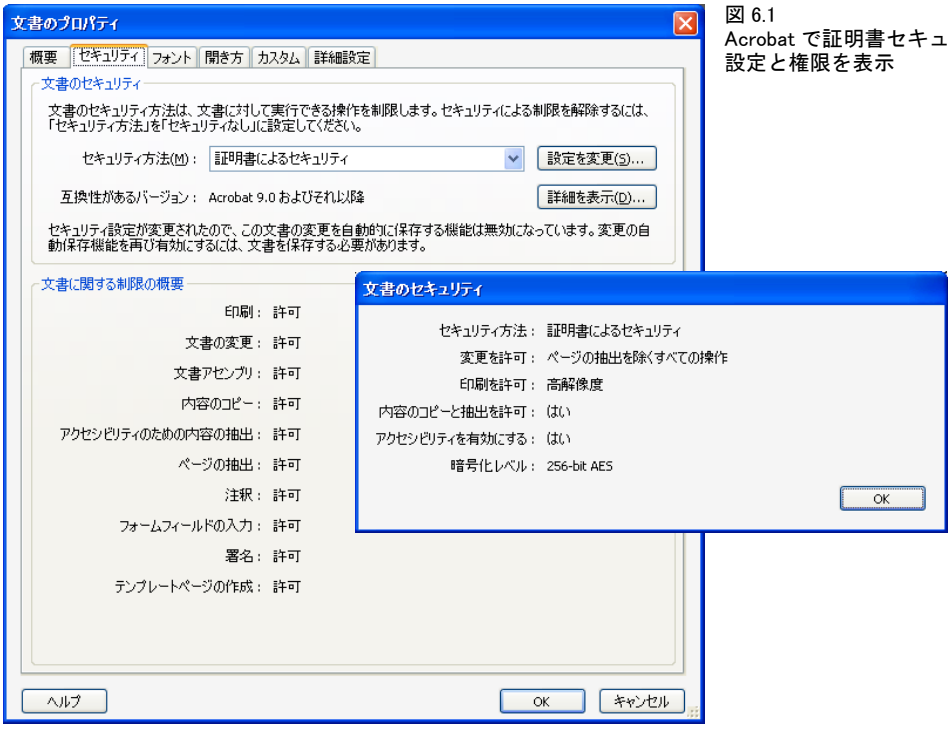


図 6.1
Acrobat で証明書セキュリティの設定と権限を表示

信者の名前がわかりますし、そのIDを悪用されてその受信者の署名の偽造に使われたりする可能性もあります。また、ハードウェアベースのデジタルIDは複製できません。

証明書セキュリティは、Acrobat・Adobe Reader 6 およびそれ以降が対応しています。以下、Acrobat における証明書セキュリティの概要を説明します。詳しくは Acrobat の説明書をごらんください。

証明書セキュリティの準備　証明書セキュリティを利用するにはデジタル証明書が必要です。もっと正確に言うと、自分のためのデジタル ID（公開鍵と秘密鍵が対になっている）と、受信者それぞれのための証明書（公開鍵のみを含んでいる）が必要です。証明書を入手する方法は大きく 2 通りあります：

- 自己署名、たとえばAcrobat を用いて生成：受信者たちから直接証明書を受け取るのなら、この方式がシンプルで、追加費用も一切かかりません。ただ、デジタルIDが紛失すると、それを取り戻すことができず、よって、暗号化された文書群をもはや開くことができなくなります。
- 商用 CA からのデジタル ID は、有料ですが、紛失の場合に取り戻せます。AATL 証明書が用いられるならば、文書に電子署名を行うためにもそれを使用できますので、Acrobat における検証は追加の構成を一切必要としません（7.1.3 節「Acrobat における信頼済みルート証明書」（93 ページ）を参照）。

保護されている文書を作成するためなら、必要なのは受信者たちの公開鍵を有する証明書だけです。一方、保護され文書を開きたいなら、各受信者だけでなく自分自身も、その秘密鍵を有するデジタル ID を持っていなければなりません。証明書は秘密情報を一切含んでいませんのでパスワードを必要とせず自由に配布できますが、デジタル ID は通常、パスワードや PIN を用いて保護されます。

復号のための自分のデジタル ID を構成　Acrobat は、保護されている文書を開くために用いることができるようにデジタル ID を構成する方法を数通りサポートしています。Acrobat XI/DCを用いて自分のデジタルIDを作成またはインストールするには次のとおり操作します：**「編集」→「環境設定 ...」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「デジタル *ID*」→「*ID* を追加」**。
すると、ダイアログ**「デジタル *ID* を追加」**が現れますので、既存の ID をファイルから追加することもできますし、新規の自己署名 ID を作成することもできます。

信頼済み証明書を書き出したいとき、または、自分のデジタル ID のための証明書を作成したい（自分のために文書を暗号化してもらえるように）ときは、Acrobat XI/DC を用いて次のとおり操作します：**「編集」→「環境設定 ...」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」**（他の人の証明書の場合）または**「デジタル *ID*」**（自分の証明書の場合）→ ID または証明書を選択→**「証明書の詳細」**。
すると、証明書ビューアが開きますので、**「書き出し ...」**をクリックし、**「書き出したデータをファイルに保存：証明書ファイル」**を選択（**「証明書メッセージシンタックス - *PKCS#7*」**ではなく）します。こうして書き出された証明書を、受信者証明書として使用することが可能です（期限が切れていないなら）。

注記 Acrobat は Windows 証明書ストア内のデジタル ID も使用できます。

Acrobat を用いて証明書セキュリティを適用　自分のデジタル ID を構成し、かつ、受信者証明書群を入手したら、Acrobat XI/DC で以下の手順で PDF 文書を暗号化できます：

- **「ファイル」→「プロパティ ...」→「セキュリティ」**タブ→**「セキュリティ方法：証明書によるセキュリティ」**をクリック。Acrobat DC では、次の手順でも可能です：**「ツー**

ル」→「保護」→「暗号化」→「証明書による暗号化」。するとダイアログが現れるので、証明書セキュリティのために使用される PDF 暗号化の種類を指定できます (図 6.2 参照)。

- 次のダイアログで、暗号化された文書を後で自分が開くことができるよう、自分のデジタル ID を選択するべきです。
- そして、任意の数の受信者証明書を選択し、権限を調整したければ調整します。受信者証明書は、Windows 証明書ストアから引き出すこともできますし、ファイルから読み取ることもできますし、オンラインレポジトリから取得することも可能です。
- ファイルを保存すると、選択したセキュリティ設定に従ってそのファイルが暗号化されます。

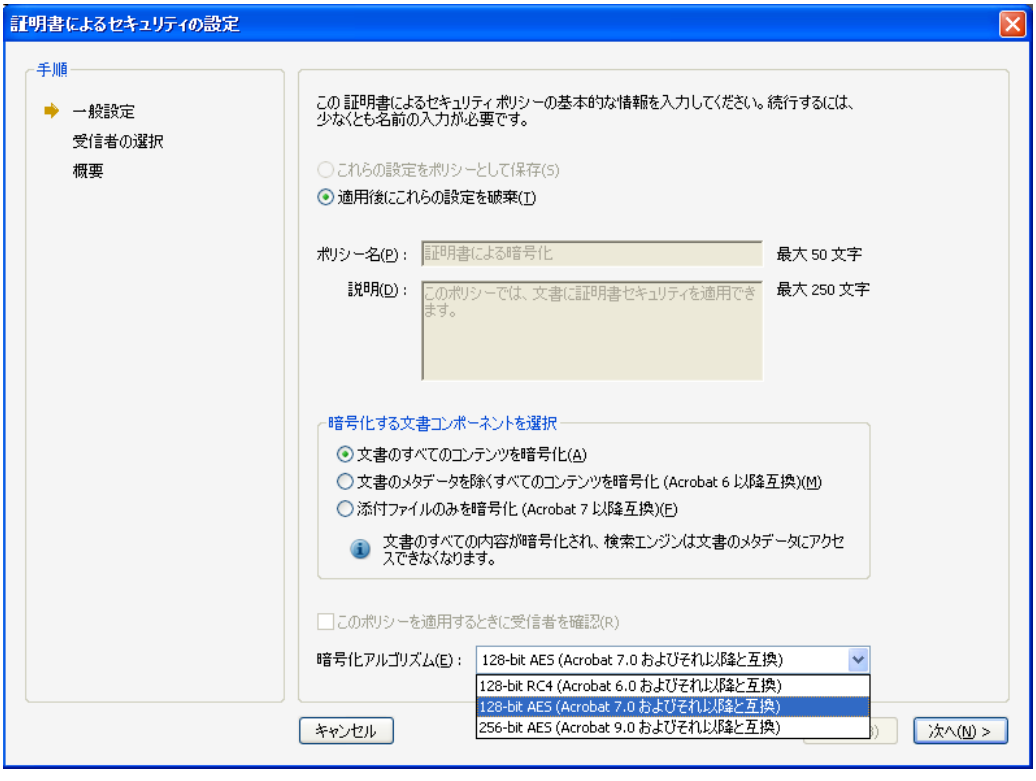
暗号化された文書を再び開くには、その受信者証明書の 1 つに照応する秘密鍵を有するデジタル ID が必要です。この ID は、Acrobat の証明書ストアにも Windows 証明書ストアにもインストールできます。

復号エラー　Acrobat は、文書を復号できないときは、以下のエラーメッセージを発します：

この文書の暗号化にはデジタル *ID* が使用されましたが、解読するデジタル *ID* がありません。デジタル *ID* が正しくインストールされているかを確認するか、文書の作成者にお問い合わせください。

ただしこのメッセージは、照応するデジタル ID が見つからないときだけでなく、文書が他の理由で復号できないときにも表示されます。

図 6.2
Acrobat の証明書セキュリティダイアログ。PDF 暗号化アルゴリズムの選択が下端にあります



ECC 受信者証明書との Acrobat の非互換　Acrobat XI 以上は、電子署名と暗号化のために、曲線 P-256/P-384/P-521 およびその他 NIST が推奨する曲線を用いる楕円曲線暗号（ECC）に対応しています。しかし Acrobat は、RFC 5753**「暗号メッセージ構文（*CMS*）での楕円曲線暗号（*ECC*）アルゴリズムの使用」**によって修正された内容の RFC 5652 に準拠していない CMS 暗号化オブジェクトを生成します。このため、暗号化された文書がサードパーティソフトウェアと互換性がなくなります。

この問題は、Acrobat DC 17.012.20093 Continuous トラックおよび Acrobat DC Classic 15.6.30352（2017 年 8 月アップデート）において修正されています。これより古いバージョンの Acrobat においては、EC 受信者証明書を用いた証明書セキュリティを避けることを推奨します

6.2 PDF における証明書セキュリティ

6.2.1 CMS 封入データ

PDF の証明書セキュリティは、RFC 5652 に従った暗号メッセージ構文（CMS）に基づいています。CMS は、電子署名・メッセージ認証・暗号化を含むさまざまな暗号機能のためのカプセル化構文を記述します。PDF の証明書セキュリティは、CMS によって提供される**封入データ**機能を利用します。なお、暗号化電子メールのしくみもこれと同様です。保管上の要請とパフォーマンスを最適化するため、証明書セキュリティはハイブリッドな手法で実装されています：まず 1 個の暗号鍵をランダムに生成し、それを各受信者の証明書に対して暗号化します。この処理は、RSA または楕円曲線暗号（ECC）アルゴリズムに基づく公開鍵暗号化によってこのランダム鍵の暗号群を生成し、CMS オブジェクト内に保管します。このランダム鍵を用いて CMS ペイロード本体を何らかの対称アルゴリズムによって暗号化し、CMS 内に保管します。対称暗号化には AES アルゴリズムを使う実装が近年は一般的です。この処理について、また、そこで使用されるさまざまなアルゴリズムについて、詳しくは 6.2.2 節「暗号化の詳細」（79 ページ）を参照してください。

受信者はおのおの、自分の秘密鍵を用いてこの対称鍵を復号し、得た鍵を用いて CMS ペイロードを復号します。この、暗号化されたペイロードと、各受信者のための暗号化された内容暗号鍵との組み合わせを、電子封筒といいます。

先に行われる暗号化は非対称であり、かつそのランダム鍵は一時的にしか保持されませんので、暗号化文書の作成者は、受信者のリストの中に自分の証明書を含めなければ、後でその文書を復号できません。

このような、非対称暗号化と対称暗号化によるハイブリッドなアプローチには、2 つの理由があります。第一に、非対称暗号化は非常に遅いので、少量のデータにしか適しません。ですのでそれを、短い対称暗号鍵にのみ適用し、ペイロード全体には適用しないのです。第二に、このアプローチであれば、ペイロードは対称暗号化によって 1 回だけ暗号化すればよく、短い暗号鍵のみを受信者ごとに暗号化すれば済みます。もしもペイロードを受信者ごとに暗号化したら、出力ファイルサイズはたいへん大きくなってしまうでしょう。

CMS オブジェクト内の *EnvelopedData* 構造は、1 個ないし複数の *RecipientInfo* 構造を内容として持ちます（図 6.4 参照）。これはそれぞれ、1 人の受信者の証明書に関する情報 — たいていは証明書発行者（CA）とシリアル番号 — と、その受信者のための暗号鍵を内容として持ちます。

受信者秘匿性　受信者の名前は CMS オブジェクト内に入っていません。ただし、証明書発行者の名前と、証明書のシリアル番号は、CMS オブジェクトから復号なしで取得が可能です。公開鍵基盤から取得した証明書の場合であれば、これによって晒されるのは CA の名前とシリアル番号のみということになります。この情報だけから受信者を特定できるかどうかは PKI によって異なります。しかし自己署名証明書の場合には、証明書保有者自身がその公開鍵に署名しています。ゆえに、自己署名証明書を用いている受信者の名前はすべて、CMS オブジェクト内でプレーンテキストで見えています。受信者情報を晒すことが望ましくない場合もあるでしょう。この問題を解決するには、自己署名証明書を使わないか、あるいは自己署名証明書では偽名を用いるとよいでしょう。

受信者数と CMS のサイズ　1 個の文書を、任意の数の受信証明書に対して暗号化することができます。ただ、受信者を追加するごとにそれぞれ個別に暗号化された鍵が埋め込まれますので、受信者が増えるほど CMS のサイズが増大します。どの程度ファイルサイズが増大していくかは、受信者の公開鍵の長さと、証明書内の情報量によって異なります。通常、出力ファイルサイズは受信者 1 人あたりおよそ 1 ～ 2 KB 増大します。

CMS を PDF 文書に適用　証明書セキュリティで暗号化されている PDF 文書は、その *Encrypt* 辞書の *Recipients* エントリの中に 1 個ないし複数の CMS オブジェクトを内容として持っています。ただし PDF は、CMS のしくみを文書内容に直接適用するのではなく、もう 1 つ暗号化の層を設けています。この暗号化はパスワードセキュリティと同じです。CMS ペイロードは、PDF オブジェクトを一切内容として持っておらず、PDF オブジェクト群のための暗号鍵を導出するために使用される鍵材料を内容として持っています。PDF オブジェクト群を暗号化するために使われる対象アルゴリズム群は、パスワードセキュリティの場合に使われるものと同じです（表 5.1 参照）。証明書セキュリティでは文書の暗号鍵をその CMS の中の暗号化された鍵材料から導出するのに対して、パスワードセキュリティではこの鍵を秘密のパスワードから導出します。

PDF 権限制限　パスワードセキュリティで行えるのと同じさまざまな権限設定を、証明書セキュリティを用いて保護される文書に適用することもできます（図 6.1 と「権限制限」（65 ページ）を参照）。パスワードセキュリティで行えるさまざまな権限設定に加えて、証明書セキュリティでは以下の設定も可能です：

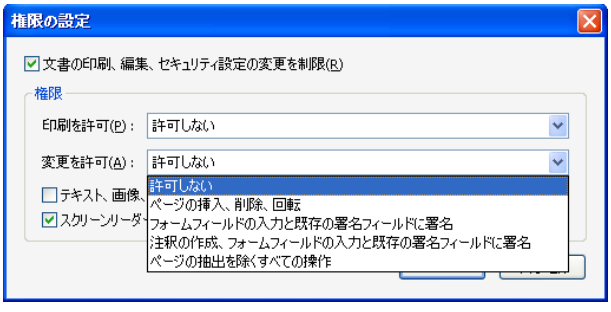
- **文書の印刷、編集、セキュリティ設定の変更を制限**（図 6.3 上端参照）：この制限が有効の場合、受信者はその文書を開いて読むことはできますが、その文書を印刷したり変更したりといった特定の操作群は、別の権限設定群によって支配されます。この制限が有効でない場合、受信者はその文書に対する完全な制御を有し、そのセキュリティ設定を変更することも可能です。これは、パスワードセキュリティを用いて保護されている文書についてユーザーがそのマスターパスワードを知っている場合と同様です。ですので、これを**マスター権限**と呼んでいます。

ユーザーごとに異なる権限設定を付与することもできます。たとえば会社の中で、上司には、その文書を編集したり暗号化を変更したり等あらゆる変更を行えるようマスター権限を付与しておき、一方で彼女の部下たちには、フォームフィールドに入力を行うこととその文書に署名を行うことのみ許可する、といった運用が可能です。受信者ごとに、あるいは受信者たちのグループごとに個別に権限を適用できることは、証明書セキュリティにあってパスワードセキュリティにはない重要な利点です。

権限は、CMS ペイロード内へ書き込まれ、ある特定の受信者のために暗号化されます。ですので、ある特定の受信者に対する権限設定を知るには、まずその受信者のための復号ができていなければなりません。受信者の権限を、照応するデジタル ID へのアクセスなしにクエリすることは不可能です。

権限の取り扱いは、PDF 文書のための証明書セキュリティと電子メールのための証明書セキュリティとの間の顕著な違いです。1 個の PDF 文書は複数の CMS オブジェクトを内容として持つことができ、各オブジェクトはさらに複数受信者のグループを指定可能で

図 6.3
Acrobat で証明書セキュリティに対する権限制限を設定



す。同一の権限を有する受信者たちのための暗号鍵群は、同じ CMS オブジェクトの中に格納されます。

6.2.2 暗号化の詳細

証明書セキュリティは複数の暗号化ステップによって実現されており、ステップごとに異なるアルゴリズムと鍵長を使用する場合もあります。これらの暗号化ステップを以下説明し、図 6.4 に示します。

ステップ 1：CMS 公開鍵暗号化と鍵ラップ　ランダムに生成された内容暗号鍵（CEK）が、公開鍵暗号化によって暗号化されます。ただしその詳細は、受信者証明書が RSA か ECC かによって異なります。RSA 鍵を用いている受信者と ECC 鍵を用いている受信者が混在していても、あるいは、異なる RSA 鍵長または ECC 曲線を用いている受信者が混在していても、差し支えはありません。

受信者の証明書が RSA アルゴリズム（RFC 5652）用の公開鍵を内容として持っている場合には、この鍵を用いて内容暗号鍵が暗号化されます。使える RSA 鍵長は、PDF Reference 内に仕様化されていませんが、Acrobat のバージョンに依存しています。RSA 暗号化はパディング方式を必須とします。デフォルトの PKCS#1 v1.5 方式は、すべての Acrobat バージョンがこれに対応しています。これよりも新しい、PKCS#1 v2（RFC 3447 と同等）・RFC 3560 に従った OAEP（最適非対称暗号化パディング）は、セキュリティ上さまざまな利点を提供します：これを要求するにはオプション *rsapadding=oaep* を用います。Acrobat DC

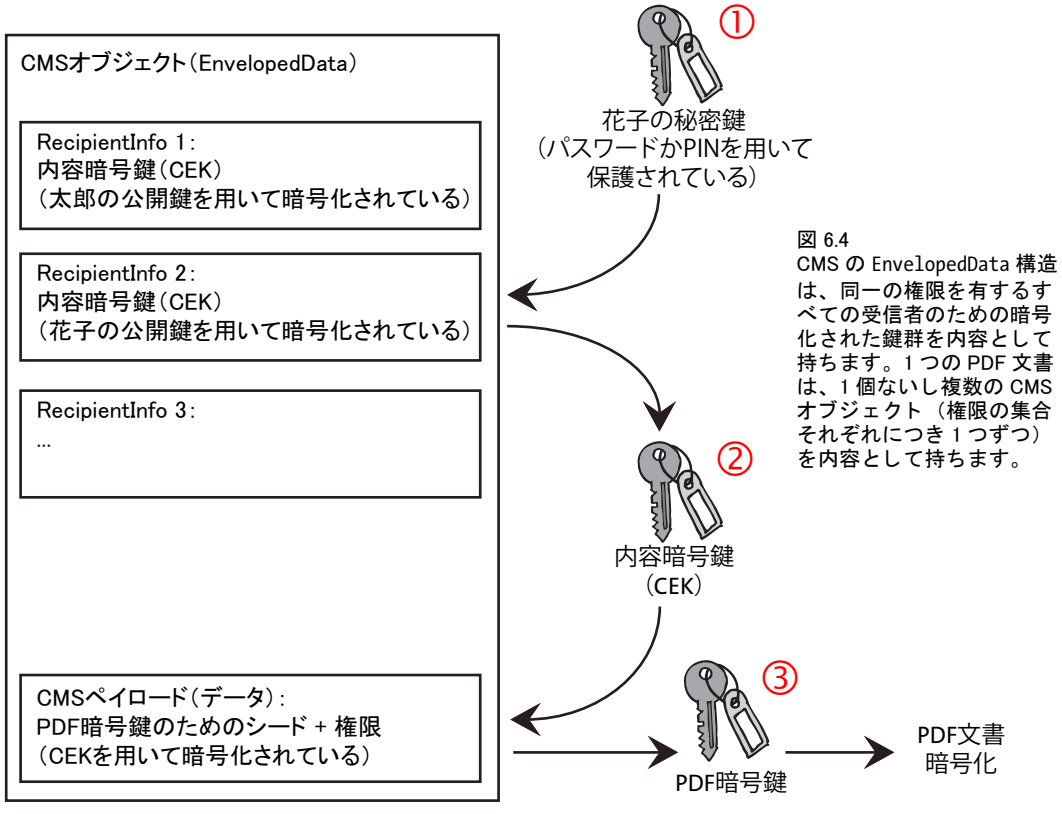


図 6.4
CMS の EnvelopedData 構造は、同一の権限を有するすべての受信者のための暗号化された鍵群を内容として持ちます。1 つの PDF 文書は、1 個ないし複数の CMS オブジェクト（権限の集合それぞれにつき 1 つずつ）を内容として持ちます。

以下は OAEP に対応していません。いくつかのサードパーティ PDF ビューアは対応しています。

受信者の証明書が公開 ECC 鍵（RFC 5753）を内容として持っている場合には、楕円曲線ディフィー・ヘルマン（ECDH）鍵共有方式とこの受信者証明書の中の公開鍵を用いてさらにもう 1 つの一時的な鍵暗号鍵が導出されます。それから、鍵ラップアルゴリズムという対称暗号アルゴリズムを使用して、この鍵暗号鍵を用いて内容暗号鍵が暗号化（ラップ）されます。Acrobat XI/DC は鍵ラップアルゴリズムとして AES-128 か AES-256 を使用します。使える ECC 曲線は、PDF Reference 内に仕様化されていませんが、Acrobat のバージョンに依存しています。

ステップ 2：CMS 内容暗号化　内容暗号鍵を用いて、対称アルゴリズムを使用して PDF 暗号鍵材料（実際の鍵そのものではなく）が暗号化され、その結果、暗号化された CMS ペイロードが生成されます。PDF の証明書セキュリティにおいては、その CMS「内容」は、PDF 文書データを一切内容として持っておらず、鍵材料を内容として持っています。この鍵材料から、PDF オブジェクト群のための最終的な暗号鍵が導出されます。

この CMS 内容暗号化アルゴリズムは、さまざまなアルゴリズムから選べます。Acrobat 7 ～ X は常に Triple-DES を使用しており、一方、Acrobat XI/DC は AES-128 を使用します。ペイロードの暗号化は、受信者数にかかわらず、1 回しか行う必要がありませんので、そのアルゴリズムの選択は受信者証明書群に依存しないのです。

ステップ 3：PDF 暗号化　PDF 暗号鍵が PDF オブジェクト群に適用され、その結果、その文書を表示するためのデータが生成されます。このステップはパスワードセキュリティと同じです。

PDF オブジェクト群を暗号化するための対称アルゴリズムと鍵長は PDF Reference 内に仕様化されており、パスワードセキュリティの場合に使用されるものの部分集合にあたります（Table 5.1 参照）。PDF 文書内のすべてのオブジェクトが、同じ対称アルゴリズムを用いて暗号化されます。ステップ 3 のためのアルゴリズムだけを Acrobat で選択できます（図 6.2 下端参照）。この PDF 暗号化アルゴリズムは、強固な暗号化を実現するために、Acrobat 7 で AES-128 が、Acrobat 9 で AES-256 が導入されています。

さまざまなアルゴリズムと鍵長　表 6.1 に、PDF・Acrobat のさまざまなバージョンで使えるアルゴリズムと鍵長をまとめます。

表 6.1 証明書セキュリティのための PDF・CMS 暗号化アルゴリズムと Acrobat での対応

PDF/Acrobat のバージョンと pCOS アルゴリズム番号	ステップ 1：CMS 公開鍵アルゴリズム	ステップ 2：CMS 内容暗号化	ステップ 3：PDF 暗号化
PDF 1.4（Acrobat 5）、pCOS アルゴリズム 5	Acrobat 6 およびそれ以降：2048 ビット RSA	Acrobat 6 ～ X：Triple-DES Acrobat XI/DC：128 ビット AES PLOP：読み取りにのみ対応	128 ビット RC4（脆弱。PDF 2.0 では非推奨）
PDF 1.6（Acrobat 7）、pCOS アルゴリズム 6	Acrobat 7 およびそれ以降：4096 ビット以下の RSA Acrobat 8 およびそれ以降：8192 ビット以下の RSA[1]	Acrobat 7 ～ X：Triple-DES Acrobat XI/DC：128 ビット AES PLOP：128 ビット AES	128 ビット AES（PDF 2.0 では非推奨）

表 6.1 証明書セキュリティのための PDF・CMS 暗号化アルゴリズムと Acrobat での対応

PDF/Acrobat のバージョンと pCOS アルゴリズム番号	ステップ 1：CMS 公開鍵アルゴリズム	ステップ 2：CMS 内容暗号化	ステップ 3：PDF 暗号化
PDF 1.7ext3（Acrobat 9）、pCOS アルゴリズム 10	Acrobat 9 およびそれ以降：8192 ビット以下の RSA[2, 3] Acrobat DC：曲線 P-256/P-384/P-521 を用いた ECC[4, 5]	Acrobat XI/DC：256 ビット AES PLOP：256 ビット AES	256 ビット AES（強固）

1. RSA-8192 鍵は、Acrobat X またはそれ以降を必要とし、macOS 用 Acrobat では対応していません。
2. Windows 証明書ストア内の ID を用いて文書を復号する場合、Acrobat は 8 ビットの倍数の長さの鍵にしか対応していません。
3. OAEP パディングを用いた RSA には Acrobat DC 以下は対応していません。
4. Acrobat DC 2017.012.20093 Continuous トラック以降が必要です。「ECC 受信者証明書との Acrobat の非互換」（76 ページ）参照。
5. 鍵導出関数 dhSinglePass-stdDH-sha512kdf（RFC 5753 においてオプショナルである）に MSCAPI は対応していません。ですのでそのような文書は PLOP で engine=mscapi を用いて復号することはできず、engine=builtin を用いてのみ復号できます。

6.3 証明書セキュリティの用途例

この節では、証明書セキュリティの利点を活用できる用途例を紹介します。多くの場合、以下の問いを分析するべきです：

- 作成者自身の証明書を受信者のリストへ含める必要があるか？ 含めない場合は、作成者はその保護された文書を開くことができなくなります。
- 各受信者に、あるいは受信者たちのグループに、どの権限制限を適用するべきか？

非公開の受信者グループへ部外秘の文書を配布　あるグループのメンバーたちが、部外秘の文書をやり取りすることによって、グループの他のメンバー全員がその文書を利用できるようにしたいと考えたとします。この PDF は、グループのメンバー全員の証明書に対して暗号化されます。文書の作成者がそのファイルを暗号化する際に自分の証明書を含めれば、その文書のバージョンは 1 つで済みます。受信者数に厳しい制限はありませんが、受信者を増やすごとに文書が少しずつ大きくなることは留意するべきです。

この用途例に似た例として、一部の受信者たち（上司たち）はその文書を変更することを許され、一般社員たちはフォームフィールドに入力を行うこととその PDF に署名を行うことだけを許されるようにしたいとします。この区別を実現するには、別々の受信者グループにして、グループごとに適切な権限を付与します。

グループのメンバー数が多くなった場合（受信者が数千名）には、そのグループをより小さな集合へ分割することもできます。各部分集合内の受信者数が少なければファイルサイズは最小化され、逆に各部分集合内の受信者数が多ければ、同じ文書を元に保護を変えて作るバージョンが少なくて済みます。

部外秘の文書に署名も行う　部外秘の文書が特定の受信者たちのために暗号化されるとします。この受信者たちは、その文書に電子的に署名を行うことが予期されていますが、変更を行うことは一切許されていないものとします。これを実現するには、その権限は署名のみを許して変更を許さないよう設定されます。受信者は、この文書を復号するために使ったのと同じデジタル ID を使って、この文書に署名を行えます。ただし、このデジタル ID が両方の操作を許可されて発行されていることが前提です。

デジタル著作権管理　商業コンテンツが有料顧客へ頒布されるとします。加入者または購入者はそれぞれ、彼個人の証明書に対して暗号化された、保護された PDF を受け取ります。同じ文書を元に保護を変えたバージョンを多数作ることによって、受信者ごとに文書の個別のバージョンを作ることができます。*nomaster* 権限制限を設定することによって、顧客が文書を改竄することを防ぎます。

セキュアなストレージとアーカイビング　このシナリオでは、アーカイブが、保護されなければならない文書を受け取るとします。アーカイブされる各文書はアーカイブ所有者の証明書に対して暗号化されます。アーカイブされる各文書の保護されたバージョンは 1 つで済みます。

インボイスと明細書の送付　顧客ごとのインボイス・明細書・取引書面は、保秘のために、その顧客の証明書に対して暗号化されます。各文書の保護されたバージョンが 1 つだけ作られてその顧客へ送付されます。*nomaster* 権限制限を設定することによって、顧客が文書を改竄することを防ぎます。

6.4 PLOP を用いた証明書セキュリティ

PDF 暗号化アルゴリズムと鍵長　PLOPは証明書暗号化を常に、強固なアルゴリズムAES-128 または AES-256 を用いて行い、脆弱な RC4 アルゴリズムを決して使用しません。この PDF 暗号化アルゴリズムを選択するには *PLOP_create_document()* の *encryption* オプションを用います：

- *encryption=algo6* の場合：PDF バージョンが必要に応じて PDF 1.6 へ上げられ、Acrobat 7（pCOS アルゴリズム 6）に従った AES-128 を用いた証明書暗号化が行われます。
- *encryption=algo10*の場合（これがデフォルトです）：PDFバージョンが必要に応じてPDF 1.7ext3 へ上げられ、Acrobat 9（pCOS アルゴリズム 10）に従った AES-256 を用いた証明書暗号化が行われます。

この PDF 暗号化アルゴリズム（すなわち AES-128 か AES-256）は CMS 内容暗号化にも使用されます。脆弱なアルゴリズムは一切使用されません。

受信者証明書を指定　受信者ごとに、その受信者の公開鍵を内容とする証明書を指定する必要があります。デジタル ID とは異なり、証明書は秘密鍵を一切内容として持っていませんので、保護される必要はありません。PDF 文書は、その指定された受信者たちだけが彼らの証明書の中の公開鍵に照応する秘密鍵を用いてそれを復号できるように暗号化されます。

出力文書を生成する前に、*PLOP_add_recipient()* への呼び出しを用いて各受信者を指定しておく必要があります（完全なサンプルコードが、すべての PLOP パッケージに同梱されている *certsec* ミニサンプルの中にあります。）：

```
if (plop.add_recipient("certificate={filename=demo_recipient_1.pem}") == -1)
{
        /* 警告を出して続ける */
        System.err.print("Warning: ", plop.get_errmsg());
}
...
if (plop.create_document(out_filename, optlist) == -1) {
        System.err.println("エラー： " + plop.get_errmsg());
        plop.delete();
        System.exit(2);
}
```

受信者を 1 人でも指定すれば証明書セキュリティが有効になります。受信者たちのリストは、ひとたび作られた後は、*PLOP_add_recipient()* へのさらなる呼び出しを用いて新たなリストが作られるまで、以後生成されるすべての文書に適用されます。

受信者たちを、PLOP コマンドラインツールで *--recipient* オプションを用いて指定することも可能です。

受信者証明書の要件　PLOP を用いて PDF 文書を保護するために用いられる受信者証明書は、以下の要件を満たしている必要があります：

- 証明書が鍵使用方法拡張を内容として持っている場合には、それがその証明書による暗号化か鍵共有を許可している必要があります。電子署名のみを許可されている証明書を用いて暗号化を行うことはできません。
- 証明書は有効でなくてはなりません。すなわち、その期限日が到来していてはいけません。

- デフォルトではRSA鍵は、その鍵長が8ビットの倍数である場合にのみ受け付けられます。半端なサイズの鍵は、オプション *conformance=extended* を用いれば受け付けられます。ただし、それによって生成される文書を、Acrobat は、Windows 証明書ストア内のデジタル ID を用いて開くことができず、Acrobat 証明書ストア内の ID を用いてしか開くことができません。64 ビットの倍数の長さの鍵を使用することを推奨します。
- P-256/P-384/P-521 以外の曲線を用いた ECC 受信者証明書はデフォルトで拒否されますが、オプション *conformance=extended* を用いれば受け付けられます。ただし、その暗号化文書は Acrobat XI/DC を用いて開くことができません。

権限制限　印刷を許可しない等の権限設定は、受信者ごとに個別に、*PLOP_add_recipient()* の *permissions* オプションで指定できます。特段指定しないかぎりすべての操作がデフォルトで許可されます。表 5.3 に、使える権限設定キーワードを挙げています。

なお、Acrobat は、文書の変更に関わる 4 種の権限制限すべてについて個別の制御を提供しておらず、いくつかの制限を一緒にまとめています。表 5.4 に、Acrobat の諸設定（図 6.3 内の「変更を許可」リストの値群）と、照応する PLOP の権限制限キーワード群のさまざまな組み合わせとの関連を示しています。

フォームフィールドに入力を行うことと署名を行うこと以外のすべての文書変更を禁止する、*PLOP_add_recipient()* に対するサンプルオプションリスト：

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

暗号化エンジン　7.2 節「PLOP DS を用いて署名する」(95 ページ) で解説している *builtin*・*mscapi* エンジンを、暗号化のための受信者証明書を取得するために使用することも可能です。

エンジンを指定するには *PLOP_add_recipient()* の *engine* オプションを用います。選択したエンジンによって、証明書選択のためのサブオプションの内容が変わります：

- *engine=builtin* の場合（デフォルト）には、証明書は PEM または DER エンコーディングの X.509 ファイルとして与えられる必要があります。例：

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

  この指定される証明書ファイルは、ちょうど 1 個の暗号化証明書を内容としている必要があります。
- *engine=mscapi* の場合には、証明書を Windows 証明書ストアから取り寄せることができます。証明書は、証明書ストアの名前と、証明書の中の受信者のサブジェクト名とによって選択されます。例：

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

保護されている文書を復号　証明書セキュリティを用いて保護されている文書を復号するには、その文書の中の受信者証明書群のうちの 1 つに照応するデジタル ID が必要です。*PLOP_open_document()* の *digitalid* オプションを、その ID にアクセスするための照応するパスワードとともに与える必要があります。例：

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

このデジタル ID を Windows 証明書ストアから取り寄せる場合には、指定されたストアの中のすべての ID が、その文書を復号するためにチェックされます。ですので、ID を選択するための *subject* サブオプションは必要ありません：

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

My はデフォルトのストア名ですので、これを以下のようにさらに省略することも可能です：

```
engine=mscapi
```

PLOP のさまざまな操作に必要な資格　PDF 文書の権限設定によって反映されている作成者の意図に厳密に従うためには、証明書セキュリティを用いて保護されている文書に対する操作をすべて許せるとは限りません。PLOP は、証明書セキュリティを用いて保護されている文書を、以下のルールに従って処理します：

- 暗号化のステータスを pCOS 擬似オブジェクト *encrypt/algorithm* 等を用いてクエリすることは、然るべきデジタル ID の有無にかかわらず、常に可能です。
- その他の文書プロパティを pCOS インタフェースを用いてクエリするには、然るべきデジタル ID が、すなわち、その暗号化文書の中の受信者公開鍵群のうちの 1 つに合致する秘密鍵を有する ID が必要です。これは *pcosmode=1* または *pcosmodename= restricted* であるかによってチェックできます。
- 文書に増分更新モードで署名する場合にも、然るべきデジタル ID が必要です。それに加えて、署名が行うことが許されるには、その文書の *noannots* 権限設定が *false* に設定されている必要があります。これは *encrypt/noannots* pCOS 擬似オブジェクトを用いてチェックできます。
- これ以外のすべての、文書に対する処理には、たとえば暗号化の除去であれ、権限設定の変更であれ、然るべきデジタル ID が必要です。それに加えて、その文書が、その文書を開くために用いられた ID に対するマスター権限を設定している必要があります。これは *pcosmode=2* または *pcosmodename= full* であるかによってチェックできます。

表 6.2 に、すべての操作に対する必要条件をまとめます。

表 6.2 暗号化された文書に対するさまざまな処理のために必要なデジタル ID

ID の有無、マスター権限、pCOS モード	pCOS を用いて暗号化ステータスをクエリ	pCOS を用いてその他の文書プロパティをクエリ	更新モードで署名する	これ以外の処理、例：暗号化を変更
なし（pCOS モード 0 ／最小）	可	不可	不可	不可
然るべき ID があり、かつ、マスター権限が設定されていない（pCOS モード 1 ／限定）	可	可	noannots=false の場合にのみ可	不可
然るべき ID があり、かつ、マスター権限がこの ID に対して設定されている（pCOS モード 2 ／完全）	可	可	可	可

pCOS を用いて受信者と権限をクエリ　pCOS 擬似オブジェクト *encrypt/recipients* を用いることによって、証明書セキュリティの有無をチェックできます。もし

```
length:encrypt/recipients
```

の値がゼロより大きいならば、この配列の各エントリは、同一の権限を有する 1 人ないし複数の受信者のグループ 1 個に対する CMS オブジェクト 1 個を内容としています。各 CMS

オブジェクトが 1 人ないし複数の受信者を内容として持ちうることから、この配列長は必ずしも受信者の総数を表しません。

pcosmode・pcosmodename 擬似オブジェクトを用いることによって、その文書を開くために然るべきデジタル ID が与えられているかどうかを、また、そのマスター権限が設定されているかどうかをチェックできます：

- 最小 pCOS モード（***pcosmode=0*** または ***pcosmodename= minimum***）：然るべきデジタル ID が与えられていない。
- 限定 pCOS モード（***pcosmode=1*** または ***pcosmodename= restricted***）：開くための然るべきデジタル ID が与えられているが、その文書はこの受信者に対してマスター権限を設定していない。その文書へ署名を行うことは、その ***noannots*** 権限が ***false*** に設定されている場合にのみ許されます。
- 完全 pCOS モード（***pcosmode=2*** または ***pcosmodename= full***）：然るべきデジタル ID が与えられており、かつ、その文書がこの受信者に対してマスター権限を設定している。文書のすべての権限が制限なく与えられており、PLOP のすべての操作が許されます。

権限制限をチェックするには、***encrypt*** pCOS 擬似オブジェクト内の以下のエントリを用います（例：***encrypt/noassemble***）：

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

なお、擬似オブジェクト ***encrypt/nomaster*** というものはありません。なぜならマスター権限のステータスは、***pcosmode=2*** または ***pcosmodename= full*** であるかによってチェックできるからです。詳しくは pCOS パスリファレンスを参照してください。***dumper*** ミニサンプルの中に、証明書セキュリティを用いている文書を識別するためのコードがあります。

6.5 コマンドラインで証明書セキュリティを適用

以下のサンプルのコマンドライン呼び出しは、長いコマンドラインオプションを用いて示されています。短縮形のオプションについては Section 3.1, ≫PLOP and PLOP DS Command-line Options≪, page 35 を参照してください。

暗号化　受信者証明書を指定するには、*--recipient* コマンドラインオプションに、あるいはその短縮形 *-r* に記します。このオプションを繰り返すことによって複数の受信者を指定することも可能です。

文書を、ただ 1 人の受信者のために、その証明書をファイルから得て暗号化：

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

文書を 1 人の受信者のために暗号化し、かつ、その権限を印刷とコピーが許されないよう制限：

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

文書を、2 人の受信者のために、それぞれの証明書をファイルから得て暗号化：

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

文書をおおぜいの受信者のために暗号化：この場合には、PLOP コマンドラインツールのための応答ファイルが役立ちます（「応答ファイル」（42 ページ）を参照）。必要なすべての受信者オプションを記述したテキストファイル *recipients.txt* を作ります：

```
--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}"
...
```

そして、この応答ファイルの名前の頭に **@** キャラクタを付けたものを PLOP コマンドライン呼び出しで与えます：

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

文書を、1 人の受信者のために、その証明書を Windows 証明書ストアから得て暗号化：

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

文書を暗号化して署名する（署名オプションについては 7.2.2 節「内蔵エンジンを用いて署名する」（96 ページ）を参照）。これは受信者証明書と署名者のデジタル ID を必要とします：

```
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
```

権限設定　文書を 2 人の受信者のために保護し、1 人目の受信者はフルアクセスを許され、2 人目の受信者はその文書にいかなる変更をも行わず署名を行うことだけを許されるようにしたいとします。権限制限キーワードが入っていませんので、フォーム記入を行うことと署名を行うことが 2 人目の受信者に許されます：

```
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
```

文書を、ただ 1 人の受信者のために、その文書を見ることだけを許されるように暗号化：

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

復号　証明書セキュリティを用いて保護されている文書を、然るべきデジタル ID をファイルから得て復号し、保護されていないバージョンを生成：

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

保護されている文書を、然るべきデジタル ID を Windows 証明書ストア ***My*** から得て復号し、保護されていないバージョンを生成。***store*** オプションのデフォルトは ***My*** であり、しかも、PLOP は然るべき ID を自動的に特定しますので、***digitalid*** オプションを省くことが可能です。秘密鍵は Windows ログインによって保護されていますので ***password*** オプションも省略できます：

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

7 PLOP DS による電子署名

注記 PDF 文書に電子的に署名する機能は、PDFlib PLOP DS でのみ利用可能であり、PLOP 基本製品では利用できません。

7.1 はじめに

7.1.1 電子署名の基本概念

電子署名について詳しく解説することはこのマニュアルの範囲を超えます。ですが、PLOP DS を用いて PDF 文書に電子的に署名する際に役割を担ういくつかの重要な構成要素について説明していきます。これらの構成要素は、全体として公開鍵基盤（Public Key Infrastructure ＝ PKI）を形成しています。

電子署名は、公開鍵暗号に基づいています。公開鍵暗号のことを非対称暗号ともいいます。これは、文書に署名した人のみが入手可能な秘密鍵と、万人がその署名を検証できるよう万人が入手可能な公開鍵とによって働きます。

証明書 公開鍵は通常、証明書というものの中に入れられて配布されます。証明書は、署名者の公開鍵と名前と連絡先を内容としています。偽造証明書を防ぐために、この情報パッケージはさらに、人物あるいはその他企業やサーバといった主体へ証明書を発行する信頼された第三者によって署名されます。このような信頼された第三者のことを、認証局（Certificate Authority ＝ CA）またはトラストセンター（TC）といいます。CA 自身の証明書のことを、ルート証明書といいます。それは多くの場合、万人がダウンロードできるように、その CA のウェブサイト上で公開されています。証明書は通常、X.509 形式で保管されます。

証明書セキュリティを用いて文書を暗号化する場合には、必要なのは公開鍵のみですので、受信者の証明書があれば足ります。一方、そのような文書を復号するには秘密鍵が必要であり、それは受信者のデジタル ID の中からのみ得られます（後述）。

証明書チェーン ある CA によって発行された署名用証明書は、その発行した CA が、または、より高次の、その中間 CA の証明書を発行した CA が、信頼できると見なされるならば、信頼できると見なされます。ルート CA から、文書に署名するために実際に使用されている最下端のエンドユーザー証明書まで、順繰りに次の証明書に署名を行うことによってつながっている証明書群のリストのことを、証明書チェーンといいます。このチェーンの中の最上位の CA の証明書のことを、ルート証明書といいます。ある署名が有効であると見なされるためには、そのチェーンの中のすべての証明書が有効である必要があります。

デジタル ID 証明書を、その証明書とそれに照応する秘密鍵の両方を内容とするパッケージと区別することは、重要です。このパッケージをデジタル ID といいます。証明書は万人に自由に配布できるのに対し、デジタル ID は注意深く保護する必要があります。なぜならデジタル ID は秘密情報（秘密鍵）を内容としているからです。電子署名を行なったり、証明書セキュリティを用いて保護されている文書を復号したりするためにデジタル ID 内の秘密鍵にアクセスするには通常、パスワードかパスフレーズが必要です。デジタル ID の保管形式として広く用いられているのは PKCS#12（Windows では PFX ともいう）です。なお、証明書とデジタル ID はいつもきちんと言い分けられているわけではありま

せん：厳密には**「デジタル *ID* を用いて文書に署名する」**と言うべきところを**「証明書を用いて署名する」**と言われることは多いです。

証明書失効確認　証明書は、ある特定の期間にわたり有効です。その失効日が過ぎれば、あるいは、その CA によって明示的に失効させられた場合には、ただちに無効になります。証明書を失効させることは、その証明書保持者が関連組織を去った、あるいは、その秘密鍵が破られたという理由で、必要となる場合があります。

証明書確認は通常、OCSP（Online Certificate Status Protocol ＝オンライン証明書ステータスプロトコル）または証明書失効リスト（certificate revocation list ＝ CRL）というプロトコルを使用したオンラインクエリの過程を経ます。両方式に関して詳しくは、「OCSP の概要」（115 ページ）・「CRL の概要」（118 ページ）を参照してください。

タイムスタンプ処理　タイムスタンプは、ある特定の時点の表現に対して電子署名を行います。この際に、その時刻は、信頼された正確な時刻情報源から取得されることができます。タイムスタンプを通常の署名の中に内蔵させることによって、ある特定の時点よりも前にその署名が、ひいてはその署名がなされた文書が存在していたことを保証することも可能です。タイムスタンプを PDF 文書に対して別途行うことも可能です。タイムスタンプサーバとプロトコルの詳細に関する詳しい情報については、7.5.1 節「タイムスタンプの構成」（121 ページ）を参照してください。

デジタル ID の取得元　デジタル ID は、さまざまな取得元から取得できます。多くの ID は、電子メールに署名を行うことを意図されています。これらの電子メール ID を用いて PLOP DS で PDF 文書に署名を行うことも可能です。デジタル ID についてどの取得元を選ぶかは、必要な ID の数（従業員ごとに 1 つずつか、それとも会社 ID 1 個のみか等）と、求める制御の程度によって異なります：

- 商用またはフリーの ID を発行しているパブリック CA のうちの一つからデジタル ID を取得します。Acrobat による署名検証を可能とするために、Acrobat 内で信頼済みルートとしてインストールされている CA からのデジタル ID を用いて署名を作成することを推奨します（「Acrobat における信頼済みルート証明書」（93 ページ）参照）。
- より大きな組織の場合：自前のプライベート CA を構築することにより、デジタル ID を自分で作成できるようにします。CA を構築するためのさまざまなソフトウェアパッケージがあります。たとえばフリーの OpenSSL ソフトウェア（***www.openssl.org*** 参照）や、Java の構成要素である ***keytool*** アプリケーション、Microsoft Windows Server オペレーティングシステムに含まれている Certificate Services 等です。
- 試験目的の場合や、制御された、または小さなユーザーグループ内でのやりとり：自己署名した証明書からのデジタル ID を作成します。Acrobat XI/DC で自己署名の証明書を作成するには以下のようにします：**「編集」→「環境設定」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「*ID* を追加」→「今すぐデジタル *ID* を新規作成」**
  その次の操作で、PKCS#12 ディスクファイルか Windows 証明書ストアをターゲットとして指定できます。PLOP DS は両方の方式に対応しています。

7.1.2 Acrobat と PDF におけるさまざまな署名

PDF では、以下に述べるさまざまな種類の電子署名を使えます。署名は PDF 内にフォームフィールドとして実装されています。PDF 署名は、常に文書全体に（個々のページにではなく）紐付いており、2 つの種類があります：

- 不可視署名は、ページ上の領域を全く占めません。これを Acrobat で表示するには**「署名」**パネルを表示させます（Acrobat XI/DC :**「表示」→「表示切り替え」→「ナビゲーションパネル」→「署名 ...」**）。
- 可視署名は、文書内のページ上のどこかに位置付けられている長方形のフォームフィールドを用います。ページ番号・フィールド名・フィールド座標を指定できます。

どちらの種類の署名に対しても、場所、署名理由、連絡先情報等、さらなる特性を指定することが可能です。

注記 以前の Acrobat のバージョンでは、有効な署名を、その署名フィールドの中に緑色のチェックマークを付けて示していました。しかし、混乱のため、また偽造の問題がありうるため（ユーザーが緑色のチェックマークを最初から付けた「視覚化ページ」を提供することが可能）、これはもう推奨される習慣ではなくなりました。1 個ないし複数の PDF 署名に対するステータス情報など署名の詳細は、Acrobat の左側の署名パネルの中に表示され、ページ上の署名長方形は変更されないままとなります（その署名の有効性にかかわらず）。これは署名のステータスをより明確にユーザーに示します。

承認署名 PDF で最も広く用いられる署名の種類は、承認署名というものです。1 個の PDF 文書は、1 個または複数の承認署名を内容とすることができます。1 個の承認署名が、種別**「署名」**のフォームフィールド 1 個の中に配置されます。このフィールドは、不可視とすることも可視とすることもできます。承認署名は、その文書がそのデジタル ID の保有者によって署名されていることを保証するとともに、文書変更が必ず検知される働きを持ちます。その文書に何か変更が加えられると、その署名は必ず無効になります。承認署名は、その署名を作成する個人か主体に紐付いています。他のなんぴとたりとも、その必要な各種証明書類へのアクセスを有しませんので、その署名者は、署名時点におけるその文書の状態を否定できません（否認防止）。

承認署名を持った文書を開く際には、Acrobat は通常、ウィンドウ上端付近に青い文書メッセージバーを表示します（ただしその文書が PDF/A に準拠している場合、またはフォームフィールドを含んでいる場合には、この署名情報よりも PDF/A ステータスメッセージかフィールドメッセージのほうが優先されて表示されます）。その署名が有効の場合には、このメッセージバーは緑のチェックマークを含みます。この署名は Acrobat の**「署名」**パネル内にも表示されます。

承認署名は、長期検証のために証明書失効情報とタイムスタンプを内容として持つこともできます。いずれのアイテムも、その署名が作成された時に、信頼されたサーバからネットワークを通じて取得されます。

承認署名は PLOP DS におけるデフォルトの署名種別です。これは少なくとも PDF 1.6 出力を必須とします。必要な場合には、PLOP DS は PDF バージョンを然るべく上げます。

承認署名は pCOS では *signaturefields[...]/sigtype=approval* として報告されます。

証明用署名 1 つの文書の中の最初の署名は、証明用署名とすることもできます。この種類を作成者署名ともいいます。なぜならこれは、その作成者がその文書を作成した通りのその文書の状態を証明するからです。この文書作成者は、その文書に対して、その署名を破ることなくある種の変更が行われることを許容することもできます。ですので証明用署名を、改変検知・防止（Modification Detection and Prevention ＝ MDP）署名ともいいます。許容する変更としては以下の種類を指定できます（表 7.6 参照）:

- 変更を一切許可しない：プレスリリースや政府刊行物といった典型的なリードオンリー文書で役立ちます。この場合には、承認または文書レベルタイムスタンプ署名を追加するだけでも、その証明書署名は無効になります。

- フォームフィールドの入力と電子署名の追加（Acrobat のメニュー項目からでなく、署名フィールドをクリックすることによる）を許可：この証明書署名は、購入注文フォーム等において、フォームユーザーが必ず真正な文書で操作を行えるようにします。ユーザーが、編集可能なフォームフィールドへの入力を行なっても、または、承認署名を行なっても、その証明書署名は無効になりません。ページをページテンプレートから産み出すことによって追加する（手動でページを追加するのではなく）ことも許容されますが、この技法が使われることはまれです。
- フォームフィールドの入力と電子署名・注釈の追加を許可：これはたとえば、公証人が、署名された文書に、その信証の性質に関する詳細を内容とする註を付けたい場合に利用できます。

証明用署名を持った文書を開いた時、Acrobat は、ウィンドウ上端付近の文書メッセージバーの中にバッジを表示します。その署名は Acrobat の**「署名」**パネル内にも表示されます（こちらも、それが有効ならばバッジとともに）。

証明用署名を PLOP DS で作成するには ***certification*** 署名オプションを用います（7.3.6 節「証明用署名」（112 ページ）参照）。証明用署名には少なくとも PDF 1.6 出力が必須です。必要ならば PLOP DS は PDF バージョンを然るべく上げます。

証明用署名は pCOS では ***signaturefields[...]/sigtype=certification*** としてレポートされます。

文書レベルタイムスタンプ署名　タイムスタンプ署名は、承認署名または証明用署名の中に埋め込まれたタイムスタンプと混同しないよう注意が必要です。1 個の文書が任意の数のタイムスタンプ署名を内容とすることができます。タイムスタンプ署名は、その文書がある特定の時点において存在していたことを確かなものとします。このタイムスタンプは、信頼されたサーバからネットワークを通じて取得され、その文書に署名した個人や主体へは関わりを持ちません。タイムスタンプ署名は長期検証のために重要な役割を果たします。なぜならタイムスタンプ署名を使って既存の署名をリフレッシュさせることができるからです。タイムスタンプ署名はフォームフィールド内に配置されますが、常に不可視です。

タイムスタンプ署名を持った文書を開いた時、Acrobat は、上端付近の文書メッセージバーの中に緑のチェックマークを表示します。その署名は Acrobat の**「署名」**パネル内にも表示されます（それが有効ならば時計とスタンプのアイコンとともに）。

タイムスタンプ署名を PLOP DS で作成するには ***doctimestamp*** 署名オプションを用います（7.5.3 節「文書レベルタイムスタンプ署名」（123 ページ）参照）。タイムスタンプ署名には少なくとも PDF 1.7ext8 出力が必須です。必要ならば PLOP DS は PDF バージョンを然るべく上げます。

タイムスタンプ署名は pCOS では ***signaturefields[...]/sigtype=doctimestamp*** としてレポートされます。

使用権限署名　1 個の文書が 2 個までの使用権限署名を内容とすることができます。これを利用すると、いくつかの編集機能を Adobe Reader で使える、いわゆる Reader 有効化された PDF 文書を作成できます。使用権限署名は、署名フォームフィールドに結び付いてはおらず、Acrobat の**「署名」**パネルに表示されません。

使用権限署名は、PLOP DS で作成することはできませんが、pCOS 擬似オブジェクト ***usagerights*** を用いてクエリすることはできます。

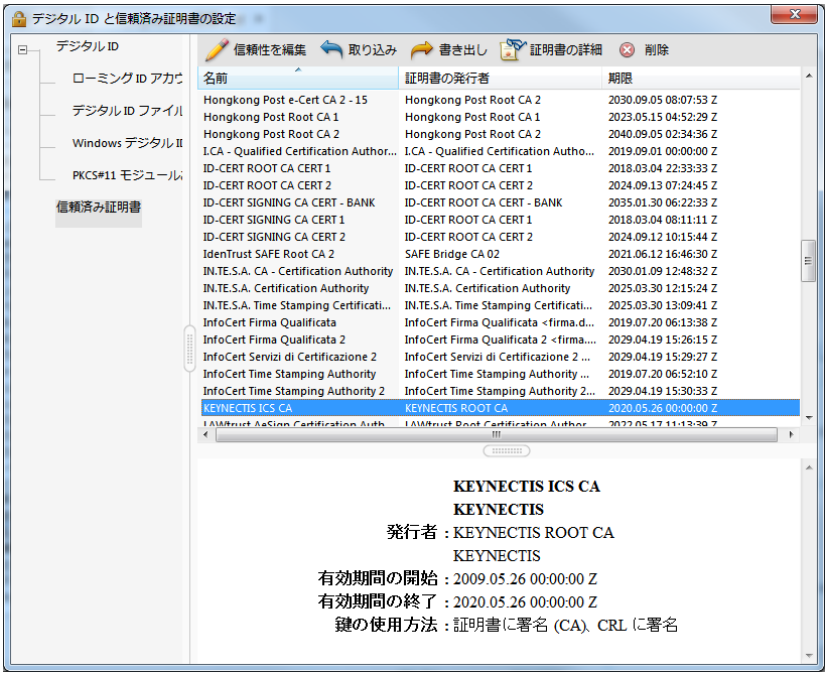


図 7.1
Acrobat の信頼済み証明書のリスト

7.1.3 Acrobat における信頼済みルート証明書

Adobe Reader と Acrobat は、以下に挙げる取得元からの CA 証明書を受け付けます。これらを信頼済みルートまたはトラストアンカーといいます。Acrobat の信頼済みルート証明書のリストを表示するには、**「編集」→「環境設定」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」**と操作します（図 7.1 参照）。信頼済みルート証明書リストの中の証明書のうちのいずれかにチェーンしている証明書は、信頼できると見なされます。Acrobat のルートストアの中の証明書のうちのいずれかにチェーンしている署名を成功裏に検証するためにエンドユーザーが Acrobat の構成を行う必要は一切ありませんので、以下の AATL または EUTL ルート CA の下の証明書を用いて署名を作成することを推奨します。

Adobe 認定信頼リスト（Adobe Approved Trust List ＝ AATL） AATL[1] は、世界中の多くの国からの商用・機関・政府認証局（CA）を内容としています。執筆時点で数十の CA が AATL プログラムに参加しています。

AATL ルート証明書群は、Acrobat・Adobe Reader X/XI/DC に内蔵されています。Acrobat は、これらのルート証明書と、遡ってこれらの信頼済みルートのうちのいずれかにチェーンしているすべての証明書を信頼します。この信頼関係を構築するために手作業で構成を行う必要は一切ありません。このリストは、定期的に自動更新させることもできますし、手作業で**「編集」→「環境設定」→「信頼性管理マネージャ」→「*Adobe Approved Trust List*（*AATL*）の自動更新」**から更新することもできます。

1. 詳しい情報と参加 CA の一覧は helpx.adobe.com/acrobat/kb/approved-trust-list1.html を参照。

AATL CA 群は、FIPS 140-2 レベル 2 に従って認証された、あるいは、EU 規則に従ったセキュア署名作成デバイス（SSCD）として認証された、ないしは同等の規格に従って認証されたセキュアトークンにのみ証明書を発行します。多くの場合その証明書は、SafeNet トークンまたはハードウェアセキュリティモジュール（HSM）に保管されます。AATL 証明書は、ファイルで配布されることは決してなく、セキュアトークンでのみ配布されます。

一部の CA は、同一のルートの下で AATL 証明書と非 AATL 証明書の両方を発行しています。この場合には、その証明書ポリシーが明示的に、その証明書が AATL の諸規則に準拠して発行されていることを宣言している必要があります。そうでなければ Acrobat がそれを、既知の信頼済みルート CA 群の下で有効と見なしません。

Acrobat は、Adobe のもっと古い認証文書サービス（Certified Document Services ＝ CDS）プログラムからの CA 証明書群も含んでいます。このプログラムは、2005 年に導入されたもので、AATL より前のものです。AATL CA は、Acrobat 内で直接、信頼済みルートとして扱われますが、CDS 証明書は、Adobe ルート証明書へチェーンしています。次の CA が CDS プログラムに入っています：Entrust・GlobalSign・Keynectis・Post.Trust・Symantec。

欧州連合信頼リスト（European Union Trust List ＝ EUTL） Adobe Reader・Acrobat XI/DC は、ETSI TS 119 612[1] に従った欧州連合信頼リスト（EUTL）からの信頼済みルート証明書群に対応しています。この EUTL の更新は、**「編集」→「環境設定」→「信頼性管理マネージャー」→「*European Union Trusted Lists*（*EUTL*）の自動更新」**から制御できます。EUTL は、規則 910/2014 に従った eIDAS フレームワークに従って、EU 全加盟国の信頼済みリスト群からのルート証明書群を含んでいます。EU のすべての国の信頼済みリストが *https://webgate.ec.europa.eu/tl-browser/#/* にあります。

信頼済みルートを手作業で追加 Acrobat は、Acrobat か Adobe Reader へ**「編集」→「環境設定」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」**から手作業で取り込まれたルート証明書も受け付けます。この証明書は、**「連絡先の信頼を設定」→「信頼」**タブから**「この証明書を信頼済みのルートとして使用」**を有効にすることによって、信頼済みとして構成される必要があります。これは、カスタムルート CA を用いる企業 PKI に対して役立つでしょう。このように構成を行う方法は、任意の信頼済みルート証明書に対して使えますが、ユーザー側による手作業を必要とするため、ワークフローによっては望ましくありません。

Windows 証明書ストア内の証明書 Acrobat は、Windows 証明書ストア内のルート証明書群を信頼済みとして扱うこともできます。これは、**「編集」→「環境設定」→「署名」→「検証」→「詳細 ...」→「*Windows* 統合」**から制御できます。

1. ec.europa.eu/digital-agenda/en/eu-trusted-lists-certification-service-providers を参照

7.2 PLOP DS を用いて署名する

7.2.1 概要

PLOP DS では、文書に電子的に署名するために必要な公開鍵とハッシュ化アルゴリズムを実装した複数の暗号化エンジンを使用できます。署名を PLOP DS ライブラリで作成するには、***PLOP_prepare_signature()*・*PLOP_create_document()*** API メソッドを用いるか、PLOP DS コマンドラインツールのオプション ***--signopt***（短縮記法：***-S***）を用います。

PLOP DS を用いて電子署名を行うにはデジタル ID が必要です。デジタル ID ファイルかトークンを使用する場合には、照応するパスワードが必要です。Windows 証明書ストア内の個人的な（アカウント毎の）デジタル ID を使用する場合には、その ID は通常、その人の Windows ログインによって保護されています。

電子署名を作成するための各種暗号化エンジン　PLOP DS ではさまざまな暗号化エンジンを使用できます。暗号化エンジンとは、電子署名を生成するために必要なさまざまな暗号化機能を実装した一片のソフトウェアまたはハードウェアです。どの暗号化エンジンを選択するかによって、デジタル ID の形式・保管場所、および他のソフトウェアとオペレーティングシステムとの統合の様式が異なります。PLOP DS では以下の暗号化エンジンを使用できます：

- ***builtin*** エンジンは、必要な暗号化機能群を PLOP DS カーネル内に直接実装しており、外部依存を一切必要としません。このエンジンはデフォルトで有効ですが、署名オプション ***engine=builtin*** を用いて明示的に選択することもできます。
- ***pkcs#11*** エンジンは、暗号トークンへの統一的アクセスを提供する PKCS#11 というソフトウェアインタフェースを参照します。ここでトークンとは、スマートカードや USB スティック等の暗号デバイスを意味します。トークンは、ソフトウェア証明書よりも高いセキュリティを実現し、多くの場合、PIN を用いて保護されています。PKCS#11 エンジンは、ハードウェアセキュリティモジュール（HSM）へアクセスするためにも使用されます。***PKCS#11*** エンジンを選択するには ***engine=pkcs#11*** 署名オプションを用います。
- ***mscapi*** エンジンは、Microsoft Cryptographic API（Windows 上でのみ利用可能）を参照します。この API はこのオペレーティングシステムに含まれています。これにより PLOP DS は、Windows によって提供されている暗号化インフラストラクチャと、あるいは、CAPI ドライバを通じて結合されているサードパーティソフトウェアまたはハードウェアと相互作用することが可能になります。***mscapi*** エンジンを選択するには ***engine=mscapi*** 署名オプションを用います。
- あるいは、ユーザーが与えた暗号化エンジンを使用することによって、すべての暗号化操作（ハッシュ化・署名）が必ず専用の暗号化ライブラリの中で実行されるようにすることも可能です。このような外部の暗号化モジュールを連携させるには、PLOP の特製のビルドが必要であり、ご要望に応じて提供します。

使用できるデジタル ID の形式　PLOP DS は、PDF 文書に署名するためにはデジタル ID を必要とします。デジタル ID は、その署名者のデジタル証明書と、照応する秘密鍵を内容として持ち、通常、パスワード等の手段によって保護されています。PLOP DS では以下の種類のデジタル ID を使用できます：

- ***engine=builtin*** を使用：PKCS#12 形式のデジタル ID ファイル（通常、***.p12*** か ***.pfx***）
- ***engine=pkcs#11*** を使用：コンピュータに接続されたスマートカード等の暗号トークン（デバイス）に保管されているデジタル ID。

► Windows で *engine=mscapi* を使用：Windows 証明書ストア内のデジタル ID。

7.2.2 内蔵エンジンを用いて署名する

内蔵エンジンはデフォルトのエンジンです。これは、ファイルベースのデジタル ID とともに動作して、完全な機能と制御を実現します。

秘密鍵を解除　デジタル ID（より正確には：デジタル ID 内に含まれている秘密鍵）は通常、パスワードかパスフレーズか PIN を用いて保護されています。なぜなら電子署名を作成するための機密である秘密鍵を内容として持っているからです。デジタル ID を PLOP DS で使用するために解錠するには、正しい認証を与える必要があります。間違ったパスワードを与えると、PLOP DS は例外を発生させます。

照応するパスワードを、*password* 署名オプションを用いて与える必要があります。PLOP DS コマンドラインツールを使用している場合には、パスワードについては、間接的に外部ファイル内で、*passwordfile* サブオプションを用いて与えることを強く推奨します。もしパスワードをパスワードファイル内でなく直接与えてしまうと、他のユーザーが読めるおそれがあります。なぜならコマンドラインはマルチユーザーシステム上では他のユーザーから見える場合があるからです。

オプションリストの例　以下の例は、PLOP DS コマンドラインツールを用いて PDF 文書に電子的に署名する方法を示します。自分のプログラムの中から署名を作成するために、*--signopt* に与えるオプションリストを、PLOP DS の API メソッド *PLOP_prepare_signature()* に与えることもできます。対応しているすべての言語バインディングのための完全なプログラミング例が PLOP DS パッケージ内にあります。これらの例では、パスワード *demo* のデジタル ID ファイル *demo_signer_rsa_2048.p12* を使用します。このファイルはディストリビューションパッケージに含まれています。

ファイル *demo_signer_rsa_2048.p12* からのデジタル ID を使用して、PDF 文書に対して不可視の署名を作成します。このデジタル ID に対するパスワードはファイル *pw.txt* の内容となっています：

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

7.2.3 暗号トークンのための PKCS#11 エンジン

PLOP DS 内の PLCS#11 エンジンを利用すると、スマートカード・USB スティック等暗号トークン上の、または、ハードウェアセキュリティモジュール（HSM）上の証明書を使用できます。そのような機器を使用して署名を作成するには、トークン個別のプロトコルを実装した DLL または共有ライブラリが必要です。この PKCS#11 DLL/SO は、そのトークンプロバイダーによって、照応するソフトウェアパッケージに含まれて提供されています。それがシステムにインストールされて、PLOP DS から利用可能になっている必要があります。Windows ではすなわち、その DLL が、Windows システムディレクトリか、PATH 環境変数内に含まれているディレクトリか、アプリケーションのカレントディレクトリへ複製されている必要があります。PKCS#11 DLL/SO が他の DLL 群に依存している場合もあることに留意してください。この場合には、そのベンダーによって提供されているすべての必須 DLL が PLOP DS から利用可能になっている必要があります。

秘密鍵を選択　1 つの署名機器が、複数のデジタル ID を内容として持っている場合もあります。たとえば 1 つは電子メールを暗号化するための物、もう 1 つは文書に電子的に署名するための物といったようにです。そのトークン上に署名証明書が 1 つだけある場合に

図 7.2
キーボード付きスマートカードリーダ（左）と暗号 USB トークン（右）。どちらのデバイスも、PKCS#11 エンジンを用いて PLOP DS へ連携できます。

は、PLOP DS は自動的にそれを選択します。そのトークン上に署名証明書が複数ある場合には、*digitalid* オプションのサブオプション *issuer*・*label*・*serial*・*subject* のうちのいずれか 1 つを与えることによって、こうした判定基準のうちの 1 つによって適切な証明書を選択する必要があります。この与えたサブオプション群がただ 1 つの署名証明書を選択しない場合には、その呼び出しは失敗し、署名は作成されません。1 つの鍵に対して、そのトークン用の管理ソフトウェアを用いて、1 つのラベルを割り当てることも可能です。発行者・シリアル番号・サブジェクトは、証明書の固有のフィールドです。

トークン上の秘密鍵を解錠　暗号トークンが、パスワードまたは PIN がソフトウェアによって与えられることを許容している場合には、*engine=builtin* の場合と同様に *password* 署名オプションを与える必要があります。そのトークンが、直接的な PIN またはパスワード入力を要求する場合（キーボード付きのスマートカードリーダ等）には、この *password* オプションを省略する（または空文字列を与える）ことができ、そのトークンのキーボードでその PIN を手入力する必要があります。パスワード／PIN 処理の詳細は暗号トークンによって異なります。

トークンによっては、一定の時間が過ぎたら、あるいは、指定された数の署名が行われたら、自動的にログアウトするものもあります。大量署名を行う場合には、そのトークンを、自動ログアウトしないよう適切に構成する必要があります。詳しくはお持ちのトークンの説明書を参照してください。トークンが自動的にログアウトすると、以下のエラーメッセージが出ます：

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

PKCS#11 のオプションリストの例　以下の例においては、ベンダー固有の PKCS#11 DLL を *cryptoki.dll* としています。実際の DLL 名はこれとは異なる可能性があります。

PKCS#11 を通じて指定されたトークンからのデジタル ID を用いて、PDF 文書に不可視署名を作成します。このトークンに対する PIN はファイル *pw.txt* の内容となっています：

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

PKCS#11 を通じて指定されたトークンからのデジタル ID を用いて、PDF 文書に不可視署名を作成します。このコマンドでは、PIN を与えていませんので、このトークンに対する PIN を、トークンに付いているキーボードで打ち込む必要があります：

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

7.2.4 ハードウェアセキュリティモジュール（HSM）のための PKCS#11 エンジン

ハードウェアセキュリティモジュール（HSM）は、秘密鍵のためのハードウェアベースのセキュリティを実現し、トークン・スマートカードよりはるかに高いパフォーマンス優位を有します。HSM は一般に以下の用途で運用されます：

- 商用 CA が HSM ベースの証明書を提供。この HSM は通常、その CA によって稼動・運用されます。要求の高い用途では、HSM が顧客の場所に置かれる場合もあります。たとえば GlobalSign・QuoVadis・Symantec が HSM ベースの AATL 証明書を提供しています。
- 企業 PKI を用いた社内 HSM 運用。
- クラウドベースの署名：HSM サービスを CPU・ストレージサービスと併せて購入できます。たとえば IBM・Amazon Web Services（AWS）・Microsoft Azure が HSM ホスティングを提供しています。

我々の試験によれば、HSM の完全なパフォーマンスは、マルチスレッドのクライアントアプリケーションか多数の独立並行クライアントによってのみ得られることがわかっています。

署名者の証明書を与える　HSM は、秘密 / 公開鍵を生成し、この秘密鍵を機器内に安全に保管します。しかし、いくつかの HSM は、署名者の証明書をその照応する公開鍵とともに保管するわかりやすい方法を（あるいは全く）提供していません。証明書は PDF 署名へ埋め込まれる必要がありますので、署名された PDF へ PLOP DS が埋め込めるように、署名者の証明書をファイルとして利用可能にしておく必要があります。これを実現するには、*digitalid* オプションの *signercert* サブオプションを用います。秘密鍵だけを保持してそれに照応する公開鍵を保持しない HSM に対しては、このオプションが必要です。この *signercert* オプションで与える証明書を、*id・label* オプションのいずれかで選択する秘密鍵に合致させることは、ユーザー側の役割です。合致していない場合には、無効な署名が生成されます。

nCipher nShield HSM（旧 Thales nShield HSM）のための PKCS#11 の例　以下の例は、nCipher nShield HSM を用いた電子署名を演示しています。前提として、この HSM を用いて 1 つの鍵ペアが生成されており、かつ、照応する証明書が手元に別個のファイルとしてあり、かつ、nShield アプライアンスに付属の管理ソフトウェアを用いて、然るべきセキュリティワールドが構成されている（詳しくは nShield の説明書を参照してください）ものとします。この状況においては、PDF 文書に以下のように署名できます：

```
plop --signopt "engine=pkcs#11 digitalid={label=demo_signer_rsa_2048 ↵
        filename={/opt/nfast/toolkits/pkcs11/libcknfast.so} ↵
        signercert={demo_signer_rsa2048.crt}}" -o signed.pdf input.pdf
```

若干入り組んだ方法を使用して、署名者の証明書を HSM 機器上に直接保管することもできますその。その状況においては、かわりに以下のコマンドを用いて PDF 文書に署名できます：

図 7.3
ハードウェアセキュリティモジュール（HSM）を PLOP DS に PKCS#11 エンジンを用いて連携できます

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

注記 別文書「nCipher e-Security PDFlib PLOP DS Integration Guide」も参照してください。

AWS CloudHSM のための PKCS#11 の例 AWS CloudHSM v2 は、機器上に秘密鍵のみを保管し、証明書の保管をサポートしていません。証明書は PDF 署名内へ埋め込む必要がありますので、署名者の証明書をファイルとして利用可能にする必要があります。これを実現するには、*digitalid* オプションの *signercert* サブオプションを用います：

```
plop --signopt "password={<username>:<pin>} engine=pkcs#11 ↵
        digitalid={filename={/opt/cloudhsm/lib/libcloudhsm_pkcs11_standard.so} ↵
        label=demo_signer_rsa_2048 signercert={demo_signer_rsa2048.crt}}" ↵
        -o signed.pdf input.pdf
```

CloudHSM で複数の PLOP オブジェクトを用いたアプリケーションをテストしていて我々は、*PLOP_prepare_signature()* がときどき以下のエラーメッセージを出して失敗することに気づきました：

```
Error in PKCS#11 operation ('couldn't open session (C_OpenSession: CKR_DEVICE_MEMORY)')
```

PKCS#11 ライブラリは、照応するコンソール出力を生成します：

```
C_OpenSession failed with error CKR_DEVICE_MEMORY : 0x00000031
```

この問題が起こるのは、前の PLOP オブジェクトをすでに削除した後に PKCS#11 ライブラリをまた読み込もうとした時です。このエラーは、CloudHSM の PKCS#11 実装の中の問題によって引き起こされていると見受けられます。この問題は、*digitalid* オプションの *sticky* サブオプションを与えることによって回避できます。

PKCS#11 セッションとマルチスレッディング 大量署名のパフォーマンスを向上させるために、PLOP DS は、PKCS#11 DLL/SO に対するロード／アンロード操作の数を最小化させ、各 PKCS#11 セッションの持続期間を最大化しています。このためには、アプリケーションは以下の条件に従う必要があります：

- いかなる時点においても、読み込める PKCS#11 DLL/SO は、そのライブラリを使用した最後の PLOP オブジェクトに対して *PLOP_delete()* が呼び出されるまで、ただ 1 つです。最後の PLOP オブジェクトを削除した後に、別の PKCS#11 DLL/SO を *PLOP_prepare_signature()* で指定できます。言い換えれば、任意の数の PKCS#11 スロットを、すべてのトークンスロットが同一の DLL/SO によって提供されている（これは通常、同一種別のトークンを意味します）限り、マルチスレッドな方式で指定することが可能です。
- ある特定のスレッドの中の *PLOP_prepare_signature()* は、別のスレッドからすでにアクセスされている PLCS#11 スロットへアクセスしてはいけません。マルチスレッドのアプリケーションで複数のスレッドの中から同一のトークンを用いて署名したい場合に

は、ミューテックス等然るべき手段によってそのスレッド群を同期させる必要があります。

- マルチスレッドのアプリケーションは、スレッドセーフな PKCS#11 ライブラリを必要とします。*digitalid* オプションのサブオプション *threadsafe=true* は、そのライブラリがスレッドセーフかどうかをチェックし、それをスレッドセーフな仕方で初期化します。
- *digitalid* 署名オプションの *sticky* サブオプションを用いると、PKCS#11 DLL/SO を、最後の使用の後にもメモリ内に保持することが可能です。これは署名操作を若干高速化する可能性があります。ただし、同一プロセス内で他の PKCS#11 DLL ／共有ライブラリをロードすることは一切できません。

ある特定のスロットに対する *PLOP_prepare_signature()* への最初の呼び出しの中で新規セッションが作成され、それと同じスレッドの中で *PLOP_prepare_signature()* が再び呼び出されるまで維持されます。あるスレッドの中でもう署名を作成しないときは、*PLOP_prepare_signature()* をオプション *signature=false* とともに呼び出すことによって、その PKCS#11 セッションを明示的に終わらせることもできます。ですので、アプリケーションは、可能な限り多数の出力文書に対して、*PLOP_prepare_signature()* を 1 回のみ呼び出すべきです。たとえば、同一の PKCS#11 スロットが指定されている限り、かつそのトークンの制約（署名の最大数や、連続署名に対する最長時間等）が満たされている限りにおいては、*PLOP_prepare_signature()* へのさらなる呼び出しは必要となりません。

大量署名を効率的に行うための完全コードサンプルが、すべての PLOP DS パッケージに含まれている *multisign* サンプルの中にあります。なお、この *multisign* のロジックは、トークンベースの署名を行ううえで、PLOP DS コマンドラインツールよりはるかに高いパフォーマンス優位を実現します。

7.2.5 Windows の MSCAPI エンジンを用いて署名する

MSCAPI エンジンを利用すると、Windows オペレーティングシステムに内蔵されている署名機能を活用できます。最も重要な点は、Windows 証明書ストア内のデジタル ID 群へアクセスできる点です。他方で、この MSCAPI エンジンには、他の暗号化エンジン群にはないいくつかの制約があります。たとえば、MSCAPI は ECDSA に対応していません。

注記 OCSP・CRL 埋め込みとタイムスタンプは `engine=mscapi` では使えません。ですので、MSCAPI エンジンでは LTV 対応署名を作成できません。

秘密鍵を解錠　その人の証明書の設定によっては、Windows 証明書ストア内のデジタル ID は、その人の Windows ログインによって保護されており、さらなるパスワードが必要ない場合もあります。その証明書を Windows 証明書ストア内へ取り込む際に高セキュリティを有効にした場合には、その証明書を使用して署名するたびに必ずそのパスワードを求められます。

MSCAPI のためのオプションリストの例　以下の例では、署名するためのデジタル ID は Windows 証明書ストア内で得られると前提しています。PLOP DS デモ証明書でこれを実現するには、ファイル *demo_signer_rsa_2048.p12* 内のデジタル ID をダブルクリックして Windows 証明書ストア内へインストールする必要があります。

Windows 証明書ストアからの（デフォルトストア *My* からの、デフォルトストア場所 *current_user* からの）証明書を使用して、PDF 文書に不可視署名を作成。これは、パスワー

ドを与える必要がないよう、そのデジタル ID がその人の Windows ログインによって保護されていることを前提としています：

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

ファイル *demo_signer_rsa_2048.p12* 内の証明書を使用して、PDF 文書に不可視署名を作成：

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

PDF 暗号化のためのマスターパスワード *SECRET* と、デジタル ID にアクセスするためのパスワード *demo* を用いて、不可視署名を作成し、文書を暗号化：

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

Windows 証明書ストアを管理　Windows オペレーティングシステムは、いくつかの証明書ストア内に編成された証明書群を保持することが可能です。PKCS#12 形式の証明書を新規にインストールするには、単にその証明書ファイルをダブルクリックして、**「証明書のインポート ウィザード」**に従います。PLOP DS パッケージ内のデモ証明書群で、パスワード *demo* を用いてこれを試すことができます。

以下のように、Microsoft 管理コンソール（MMC）を用いて証明書群を表示・編成することもできます：

- **「スタート」**をクリックし、プログラム名のための枠に**「*mmc*」**と打ち込むことによってこのプログラムを起動します。
- **「ファイル」**メニューで**「スナップインの追加と削除 ...」**をクリックします。
- **「利用できるスタンドアロン スナップイン」**で**「証明書」**を選択して**「追加」**をクリックします。
- その次のダイアログで**「ユーザー　アカウント」**と**「完了」**を選択します。あるいは、**「サービス アカウント」**か**「コンピューター アカウント」**が自分の証明書のストア場所であるならば、それを用います。
- **「*OK*」**をクリックします。

これで、インストールされた証明書をブラウズできるようになりました。自分の証明書は**「個人」**カテゴリ内にあり、これを PLOP DS で以下のオプションリストを用いて（*--signopt* コマンドラインオプションか *PLOP_prepare_signature()* に与えて）指定できます：

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

証明書の詳細を表示するには、MMC 内で証明書をダブルクリックします。証明書を PFX 形式で書き出すには、リスト内の証明書を右クリックして**「すべてのタスク」→「エクスポート ...」**をクリックします。すると、**「証明書のエクスポート ウィザード」**が起動します。

管理コンソールを使用して証明書を取り込むこともできます：証明書ストア（**「個人」**等）を右クリックして、**「すべてのタスク」→「インポート ...」**を選択します。

7.2.6 暗号化の詳細

電子署名は、暗号化アルゴリズムと、ハッシュアルゴリズムと、両者に対するパラメータ群によって特徴付けられます。署名を生成するための暗号化アルゴリズムと鍵長は、その署名者のデジタル ID によって決定されます。それらは、そのデジタル ID のための公開鍵

／秘密鍵ペアを作成する際に指定されます。PLOP DS は以下の署名アルゴリズムに対応しています：

RSA 署名　範囲 1024 ～ 8192（3072 ビット以上推奨）の鍵長を持った RSA に対応しています。RSA は、インターネット等多くの応用分野で広く利用されています。RSA 署名は符号化方式というものを必要とします（*Encoding Method for Signatures with Appendix ＝ EMSA*）：

- PKCS#1 v1.5 に従ったデフォルト符号化方式は、Acrobat の全バージョンがこれに対応しています。ただし、多くの署名アプリケーションにおいてこれはフェーズアウトしつつあります。
- より新しい、RFC 3447/RFC 8017 に従った EMSA-PSS（*Probabilistic Signature Scheme*）符号化方式は、証明可能なセキュリティを提供します。EMSA-PSS は、SSA-PSS または PKCS #1 v2.1 とも呼ばれます。EMSA-PSS 署名を生成するには、署名オプション *rsaencoding=pss* を用います。
  EMSA-PSS の検証には、Windows 版の Acrobat XI 11.0.19 か Acrobat DC 2015.006.30280（2017 年 1 月アップデート）以降か、macOS 版の Acrobat DC 2015.006.30392 が必要です。これらより古いバージョンの Acrobat は PSS 署名を無効と見なします。Acrobat を用いた PSS 署名の生成は、macOS と Windows の Acrobat DC 2017.012.20093 Continuous トラック（2017 年 8 月にリリースされた）で動作します。これらのバージョンの Acrobat DC は、RSA 符号化方式を**「署名の詳細プロパティ」**ダイアログに表示もします（図 7.4 参照）。

注記　EMSA-PSS 署名は `engine=mscapi` に対してはサポートされていません。

DSA 署名　範囲 1024 ～ 4096（3072 ビット以上を推奨）の鍵長を持った DSA に対応しています。DSA は広く利用されていません。安全でない SHA-1 ハッシュアルゴリズムを用いた DSA にしか Acrobat は対応していませんので、DSA の使用については安全性の懸念があります。

注記　*engine=pkcs#11* に対する DSA 署名には対応していません。

楕円曲線暗号　ECDSA（楕円曲線電子署名アルゴリズム）は、RSA の近年の後継者です。鍵長は通常、512 ビット以下です（256 ビット以上を推奨）。ECDSA が RSA より優れている点は、より小さい鍵サイズで同じ暗号強度を実現できることであり、したがってパフォーマンス上の優位が期待できる点です。ECDSA の強度は曲線によって決まります。曲線は、パラメータ群によって特徴付けられますが、名前で特徴付けられることのほうが多いです。ECDSA 曲線には、広く利用されている 3 種のグループがあります：

- 最も広く利用されている曲線群は、NIST によって標準化されたもので、RFC 5480 でリストされています。これらを ***P-256・P-384・P-521*** といい、別の名前で ***secp256r1***（または ***prime256v1***）・***secp384r1***・***secp521r1*** ともいいます。Acrobat XI/DC はこれらの曲線に対応しています。
- RFC 5480 はこの他にも、NIST によって勧告された 12 種の名前付き曲線を定義しています。Acrobat XI/DC は、デジタル ID を Acrobat に直接読み込んだ場合にはこれらの曲線にも対応しますが、Windows 証明書ストアまたは macOS キーチェーンの中の証明書に対してはこれらの曲線に対応していません。
- RFC 5639 は、Brainpool 曲線という曲線群の集合を定義しています。Brainpool 曲線に基づいた署名を Acrobat XI/DC を用いて検証することはできません。残念ながら Acrobat は、この署名アルゴリズムに対応していないということを明確に示さず、Brainpool 署名に対して以下のエラーメッセージを発します：

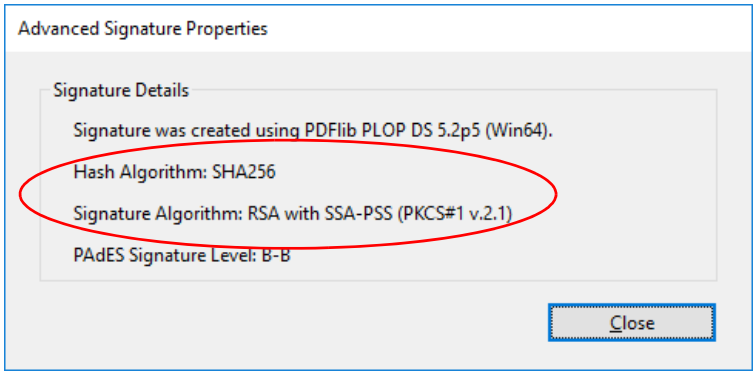


図 7.4
署名のハッシュアルゴリズムと符号化方式を Acrobat で表示

```
There are errors in the formatting or information contained in this signature.
```
（＝この署名のフォーマットまたは情報には誤りがあります。）

Acrobat は Brainpool 曲線に基づく署名を検証できませんので、これらの曲線は署名オプション *conformance=extended* を必要とします。

注記 *engine=mscapi* に対する ECDSA 署名には対応していません。

さまざまなハッシュアルゴリズム　ハッシュアルゴリズムは、署名されたデータに対するメッセージダイジェストを生成するために使用されます。広く用いられているハッシュアルゴリズムは、SHA-1（もはや安全とは見なされていません）と、それよりも強固な、SHA-2 ファミリ内の SHA-256・SHA-384・SHA-512 を含むアルゴリズム群です。署名に対して用いられているハッシュアルゴリズムを Acrobat XI/DC で表示するには以下のようにします（図 7.4 参照）：

- **「署名」**パネルを開きます。
- 署名を選択し、**「署名」**メニューで**「署名のプロパティを表示 ...」**を選択します。
- **「詳細プロパティ ...」**をクリックします。
- すると、**「署名の詳細プロパティ」**というタイトルのダイアログが表示され、その中の**「署名の詳細」**の中にハッシュアルゴリズムと符号化方式が表示されます。

表 7.1 に、各種署名アルゴリズムと、照応するハッシュ関数を挙げます。この表には、署名を検証するために最低限必要な Acrobat バージョンも挙げてあります。Acrobat を用いて PDF 署名を検証しようとする場合には、必ずその署名の諸特性に対応しているバージョンの Acrobat を使う必要があります。表 7.1 には、各署名アルゴリズムから生成される最低限の PDF 出力バージョンもあわせて挙げてあります。入力文書がそれよりも低い PDF バージョン番号を用いている場合には、PLOP DS は、その出力文書の PDF バージョンを、この表に挙げてある値まで上げます。

表 7.1 署名アルゴリズム・ハッシュアルゴリズム・PDF 出力バージョン・必須 Acrobat バージョン

署名アルゴリズム	ハッシュアルゴリズム	PDF 出力バージョンと、検証のために必要な最小 Acrobat バージョン[1]
承認・証明用署名		
8192 ビット以下の RSA	SHA-256	sigtype=cades：PDF 1.7ext8 ／ Acrobat X sigtype=cms：PDF 1.6 ／ Acrobat 7[2] rsaencoding=pss：Windows 版の Acrobat XI 11.0.19 か Acrobat DC 2015.006.30280（2017 年 1 月）。macOS 版の 2017 年 7 月アップデート
4096 ビット以下の DSA	SHA-1（Acrobat X/XI/DC は DSA に対してこれ以外のハッシュアルゴリズムに対応していませ）	sigtype=cades：PDF 1.7ext8 ／ Acrobat X sigtype=cms：PDF 1.6 ／ Acrobat 7
NIST 曲線 P-256/P-384/P-521 を用いた ECDSA（RFC 5480）	その曲線の強度によって、SHA-256・SHA-384・SHA-512 のいずれか	PDF 1.7ext8 ／ Acrobat XI
P-256/P-384/P-521 以外の NIST 曲線を用いた ECDSA（RFC 5480）	その曲線の強度によって、SHA-256・SHA-384・SHA-512 のいずれか	PDF 1.7ext8 ／ Acrobat XI/DC（Windows 証明書ストアまたは macOS キーチェーンでは不可）
14 種の Brainpool 曲線を用いた ECDSA（RFC 5639）	その曲線の強度によって、SHA-256・SHA-384・SHA-512 のいずれか	PDF 1.7ext8 ／ Acrobat XI/DC を用いて検証できません（conformance=extended が必要）
文書レベルタイムスタンプ		
TSA によって決定	デフォルトで SHA-256、ただし doctimestamp サブオプション hash を用いて変更可能	PAdES パート 4 拡張を伴う PDF 1.7ext8 ／ Acrobat X
OCSP 要求・応答（証明書識別）		
OCSP レスポンダによって決定	デフォルトで SHA-1、ただし ocsp サブオプション hash を用いて変更可能	Acrobat DC およびそれ以前は OCSP に対して SHA-1 にのみ対応（それ以外なら conformance=extended が必要）

1. PDF/A・PDF/X モードでは、入力文書の PDF バージョンが変更されないまま保たれます。
2. RSA-8192 鍵は、検証のために Acrobat X またはそれ以降を必要とし、また、macOS 上の Acrobat では対応していません。

7.3 PDF の署名の各種設定内容

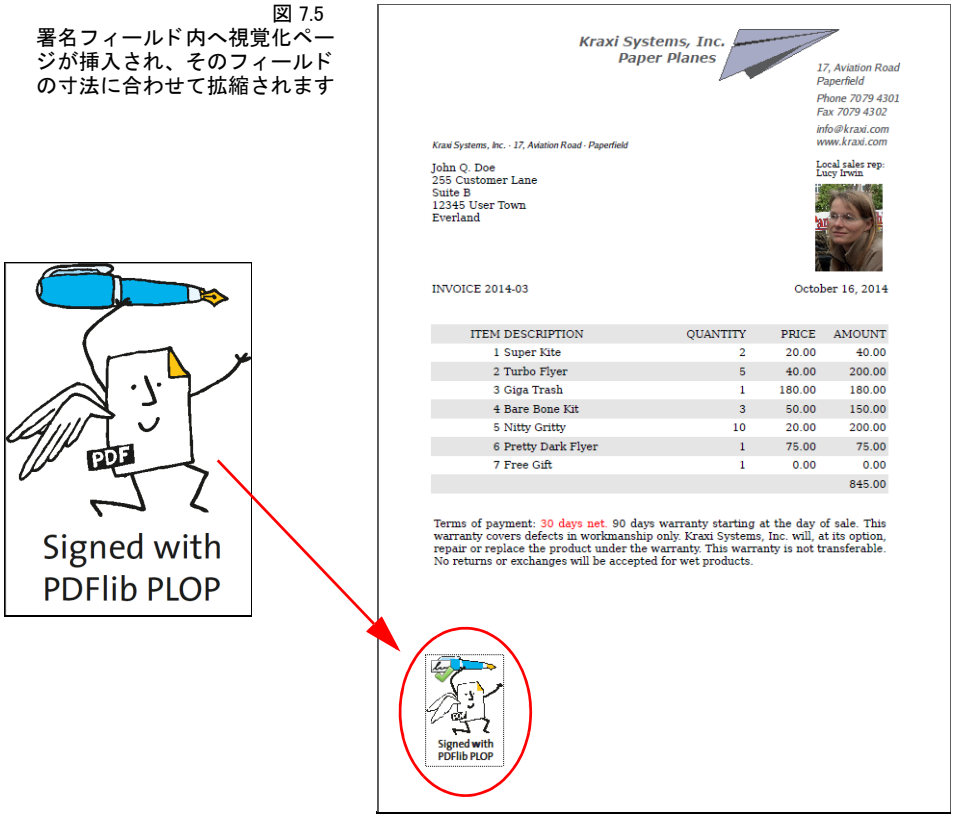
7.3.1 グラフィックかロゴを用いて署名を視覚化

電子署名は文書内へ以下の方式で組み込めます：

- 不可視署名は、ページ上に何ら表現を持ちません。Acrobat の**「署名」**パネル内にのみ表示されます。文書レベルのタイムスタンプ署名は常に不可視署名として生成されます。
- 可視署名は、ページ上の特定位置にその署名を表示するための任意のテキストかグラフィックを内容とすることができます（図 7.5 参照）。既存の PDF 文書からのページを用いてその署名の視覚表現を作成することができます。可視署名は**「署名」**パネル内にも示されます。このページを採られた文書を視覚化文書といいます。その署名を保持しているフィールドの中にこの視覚化ページが配置されていますので、Acrobat で、署名された文書の中の視覚表現をクリックすれば、その署名を検証できます。

注記 技術的には、この視覚表現を複数のページで繰り返すことも可能でしょうが、PLOP DS はこれをサポートしていません。なぜなら、唯一性のない署名視覚化には、法的な不透明性がいくつもあるからです。これらの不透明性があるので、PDF 2.0 では、署名視覚化の繰り返しを明示的に禁じています。

図 7.5
署名フィールド内へ視覚化ページが挿入され、そのフィールドの寸法に合わせて拡縮されます



署名視覚化文書　署名視覚化のために用いられる PDF ページは、スキャンされた手書きの署名や、公的印鑑や企業ロゴや、署名用証明書の保有者の写真等、その署名された文書の受け手にとって役立ちうる任意の視覚表現を内容とすることができます。

視覚化文書が、署名された入力文書よりも高い PDF バージョンを用いている場合には、生成される出力の PDF バージョンは然るべく調整されます。*PDF 1.7ext3*（Acrobat 9）と *PDF 1.7ext8*（Acrobat X/XI/DC）の文書は、その視覚化ページとしての用途に関する限り、PDF 1.7 と互換です。

注記 PDF/A のための署名視覚化は、その視覚化文書にいくつか特定の制限を課します（「PDF/A 準拠」（107 ページ）を参照）。PDF/X・PDF/VT モードでは電子署名の視覚化に対応していません。

視覚化文書を、*PLOP_open_document()* を用いて開く必要があります。その文書ハンドルを、*field* オプションの *visdoc* サブオプションに与える必要があります：

```
field={visdoc=< ハンドル > rect={100 100 300 150}}
```

署名フィールドの位置と寸法　*field*署名オプションは、ページ上でのその署名の表現を制御します。署名された文書の可視ページ上での署名視覚化ページの位置と寸法を、この *field* オプションの *rect* サブオプションを用いて指定できます。この寸法を、明示的に指定することもできますし、1 つの隅と、他の寸法のうちの 1 つか 2 つを指定することによって暗黙的に指定することもできます。抜けている値は、キーワード *adapt* を用いて指定することによって、変倍がかからないよう自動的に算出されます。この *adapt* キーワードを用いれば、視覚化ページを署名長方形の任意の隅に寄せることができます。できあがる長方形がページをはみ出してはいけません。以下の例でさまざまな組み合わせを演示します：

- 最もシンプルはアプローチは、視覚化ページを、求める寸法で作成しておくことです。この場合には、単にフィールドの左下隅の座標を与えれば、PLOP DS はページの原寸を用いて署名視覚化を行います：

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

- 左下隅に寄せて、幅を保ち、高さを調節することによって変倍を防ぎます：

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

- 左下隅に寄せて、幅を調節して高さを保つことによって変倍を防ぎます：

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

- ページを長方形に強制的に合わせます。すなわち、長方形の幅と高さを両方とも保ちます。ページと長方形の縦横比が異なる場合には、その視覚化ページには変倍がかかります：

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

然るべき署名フィールド長方形を、視覚化ページの寸法に応じて動的に算出するには、pCOS インタフェースを用いてそのページ寸法をクエリすることができます（pCOS ページ番号は 0 から始まることに留意してください）：

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

既存のフォームフィールド内に署名する　入力文書がすでに署名フィールドを内容として持っている場合には、そのフィールドをその署名と視覚化のために使用することもできます。これを実現するには、その既存のフィールドの名前がわかっている場合には、それを与えます：

```
field={name=MyExistingFieldName visdoc=< ハンドル >}
```

そのフィールド名がわからない場合には、以下のように、PLOP に対して、既存の署名フィールドを使用するよう命令することもできます：

```
field={fillexisting visdoc=< ハンドル >}
```

既存のフィールド内に署名する場合であっても、*rect* フィールドオプションを用いてその位置と寸法を変更することは可能です。既存のフィールド内に署名を作成した場合、かつ、そのフィールドがページ上の可視の長方形を用いている場合には、*visdoc* オプションを与える必要があります（あるいは、フィールドオプション *rect={o o o o}* を用いてそのフィールドを不可視にします）。

署名フィールド内における視覚化ページの位置を指定　視覚化ページは、署名フィールド内に配置され、それがその長方形内に収まり、かつその縦横比が保たれるように拡縮されます。これはとりわけ、署名を既存のフォームフィールド内へ配置したい場合、かつ、そのフィールドと視覚化ページの縦横比が一致しない場合に役立ちます。

field オプションの *position* サブオプションを用いると、その署名フィールド内における視覚化ページの位置を指定することができます。

デフォルトでは、視覚化ページはフィールド内で縦横ともに中央に配置されます。これを変更して、たとえば視覚化ページを署名フィールドの左下隅に配置させることもできます：

```
field={name=MyExistingFieldName visdoc=< ハンドル > position={left bottom} }
```

pCOS　署名の可視性は pCOS において *signaturefields[...]/visible=true* として報告されます。署名フィールドがすでに署名を内容として持っているかどうかの情報をクエリするには *signaturefields[...]/sigtype != none* を用います。

7.3.2 PDF/A・PDF/UA・PDF/X・PDF/VT 準拠

このマニュアルで別途特記しない限り、すべての PLOP 操作は、PDF/A・PDF/UA・PDF/X・PDF/VT の諸規定に準拠していますので、PLOP によって規格準拠は維持されます。ただしこの規則にはいくつか例外として、ある特定の規格によって PLOP 操作が禁じられている場合もあります。たとえば PDF/A で暗号化は禁じられています。そのような場合には、自分の優先順位を考慮する必要があります：

- 規格準拠を維持する必要がある場合には、その操作は PLOP によって拒絶されます。これがデフォルト動作です。
- その操作（暗号化等）が規格準拠よりも大切である場合には、*sacrifice* オプションを用いて規格識別子を除去することが可能です。

関連する規格ごとの注意点を以下に挙げます。

PDF/A 準拠　PDF/A 規格は、CMS・CAdES ベースの署名を許容しています。PDF/A-2・PDF/A-3 は、タイムスタンプと失効情報、および、証明書チェーンのうち入手可能な限

り多くを埋め込むことを推奨していますが、これは厳格な要求ではありませんので、PLOP DS によって強制されません。

PDF/A モードでは、すなわち、入力文書が PDF/A に準拠しており、かつ、*pdfa* に対して *sacrifice* オプションが設定されていない場合には、署名視覚化文書は、その PDF/A 諸特性について互換である必要があります：

- 視覚化文書の PDF/A レベルが互換である必要があります（表 7.2）。
- 視覚化文書の出力インテントが互換である必要があります（表 7.3）。

備考：出力インテントを持たない PDF/A-1a 視覚化文書（表 7.2 と表 7.3 で赤で囲ってあります）は、すべての PDF/A パート・互換レベル・出力インテント種別と互換です。PLOP DS ディストリビューションは、これらの特性を持ったサンプル視覚化ファイル *signing_man_pdfa1a.pdf* を含んでいます。これを、すべての種類の PDF/A とともに試すための視覚化文書として利用できます。出力インテントを持たない PDF/A-1b 視覚化文書は、すべての PDF/A パートの *b* 準拠レベル群と互換です。

PDF/A 準拠にこだわらない場合には、以下のオプションを用いてその規格準拠エントリを除去することもできます：

```
sacrifice={pdfa}
```

表 7.2 さまざまな PDF/A 入力レベルに対して互換な視覚化文書の PDF/A レベル

	視覚化文書の *PDF/A* レベル				
入力文書の PDF/A レベル	**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	許容	—	—	—	—
PDF/A-1b:2005	許容	許容	—	—	—
PDF/A-2a・PDF/A-3a	許容	—	許容	—	—
PDF/A-2b・PDF/A-3b	許容	許容	許容	許容	許容
PDF/A-2u・PDF/A-3u	許容	—	許容	—	許容

表 7.3 視覚化文書の PDF/A 出力インテント互換性（すべての PDF/A 互換レベルに対して）

	視覚化文書の出力インテント種別			
入力文書の出力インテント種別	**なし**	**グレースケール**	**RGB**	**CMYK**
なし	許容	—	—	—
グレースケール *ICC* プロファイル	許容	許容[1]	—	—
RGB ICC プロファイル	許容	—	許容[1]	—
CMYK ICC プロファイル	許容	—	—	許容[1]

1. 視覚化文書の出力インテントと入力文書の出力インテントとが同一である必要があります。

PDF/UA 準拠 PDF/UAの不可視署名フォームフィールドに対する要件は、「タグ付きPDFベストプラクティスガイド」（PDF 協会の PDF/UA 技術センターが 2019 年に発行）において緩和されています。とりわけ、不可視署名フィールドを構造ヒエラルキー内に含める必要がなくなり、特殊な準備または署名オプションが一切必要なくなりました。

注記 **Acrobat DC のアクセシビリティチェックはこの緩和された規則を実装していません。不可視署名フォームフィールドが構造ヒエラルキー内に含まれていない場合には Acrobat DC は依然として「Tagged annotations – Failed」と報告します。**

可視署名フィールドを使用するには、入力文書内で然るべきフォームフィールドを代替テキストとともに準備する必要があります。新規フィールドを作成することはできないのです。Acrobat XI/DC でこれを既存の PDF/UA 文書に対して行うには以下のように操作します：

- Acrobat DC：**「ツール」**→**「フォームを準備」**をクリック→文書を選択→「開始」。上端近くのツールバーの中のフォームツール群の中で「署名」フォームツールを選択します。
  Acrobat XI：**「ツール」**パネルを開き、**「フォーム」**パネルを開きます。**「作成」**を選択します。現れたダイアログで、**「既存の文書から」**→**「次へ」**→**「現在の文書」**を選択します。**「フォーム」**パネルの**「タスク」**セクション内で、**「新しいフィールドを追加」**→**「電子署名」**をクリックします。
- ページ上にフォームフィールド長方形を描きます。
- 「フォームを準備」ウィンドウを閉じて（Acrobat DC）、または**「フォームの編集を閉じる」**をクリックして（Acrobat XI）、**「タグ」**パネルを開きます。
- この**「タグ」**パネルの上端にあるオプションボタンをクリックして、**「検索 ...」**を選択します。
- 現れたダイアログで、**「マークされていない注釈」**を選択し、**「検索」**をクリックします。
- 作成したばかりの署名フィールドがハイライトされていることを確認してください。**「エレメントを検索」**ダイアログで**「タグエレメント」**をクリックし、**「種別：*Form*」**を選択します。ここでフィールドタイトルを与えることもできます。そして**「*OK*」**をクリックします。
- **「タグ」**パネル内で、新たに作成された *Form* 構造エレメントが、タグリストの末尾に現れているはずです。そのタグを選択し、それをタグヒエラルキー内の、その署名フィールドが読み取られてほしい位置に照応する、構造ツリー内の然るべき位置へ移動させます。
- 署名フィールドには代替テキストを割り当てることを推奨します：ヒエラルキー内の *Form* 構造エレメントを右クリックし、**「プロパティ ...」**を選択し、そのフィールドのための然るべき代替テキストを入力します。

署名フィールドに *Signature1* という名前が付けてあるとして、署名のためのターゲットフィールドとして、署名オプションリスト内でそれを名前で参照できます：

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

あるいは、既存のフィールドの中へ、その名前にかかわらず、署名を配置するよう PLOP DS に命じることもできます：

```
field={fillexisting}
```

field 署名オプションの *tooltip* サブオプションを用いると、その署名フィールドの、スクリーンリーダソフトウェアによって使われるための、然るべき代替説明を与えることができます。

PDF/UA 準拠にこだわらない場合には、以下のオプションを用いてその規格準拠エントリを除去することもできます：

```
sacrifice={pdfua}
```

PDF/X・PDF/VT 準拠　PDF/X・PDF/VT モードでは署名視覚化には対応していません。

PDF/X 準拠にこだわらない場合には、以下のオプションを用いてその規格準拠エントリを除去することもできます（PDF/VT についても同様です）：

```
sacrifice={pdfx}
```

7.3.3 文書セキュリティストア（DSS）

文書セキュリティストア（Document Security Store ＝ DSS）という専用の PDF データ構造は、証明書と、関連する OCSP・CRL 失効情報を保持することができます。このデータを、まとめて検証情報といい、長期検証のために重要な役割を果たします。この DSS は、PAdES パート 4 で導入されたものであり、ISO 32000-2 に盛り込まれています。この DSS は、認証・証明用署名に対してはオプショナルですが、文書タイムスタンプとタイムスタンプ付き署名の長期検証を可能にするには必須です。

署名オブジェクト内でなく DSS 内に検証情報を保管すると、ファイルサイズを削減できます。なぜなら署名オブジェクトと異なり、DSS は圧縮することが可能であり、かつ ASCII 表現（署名のサイズを倍増させる）を必要としないからです。さらに、DSS は、複数の文書署名を検証するためのデータを保持することが可能ですが、署名オブジェクトは、ただ 1 個の署名のための検証情報しか保持できません。

検証情報のなかには、署名オブジェクト内にしか保管できない項目もあり、DSS 内にしか保管できない項目もあり、どちらにも保管できる項目もあります。どちらにも保管できる項目については、署名オプション ***dss*** を用いると、その保管場所を制御できます。この 2 種類の場所の比較を表 7.4 に挙げます。

表 7.4 検証情報のさまざまな項目の保管場所

	署名オブジェクト	文書セキュリティストア（DSS）	dss オプションによる制御
署名用証明書	可	—	—
TSA 証明書	—	可	—
署名用証明書・TSA 証明書以外の証明書（証明用証明書の発行者等）、および照応する OCSP 応答と CRL	可	可	可
署名用証明書のための OCSP 応答と CRL	可	可	可
TSA 証明書のための OCSP 応答と CRL[1]	—	可	—

1. タイムスタンプのための検証情報を埋め込む必要がある場合には、PLOP DS は常に、DSS を増分更新として追加します。

PLOP DS は、以前の署名のための検証情報を持った既存の DSS が入力文書内にあれば、それを温存します。新しい DSS は、既存の DSS の内容に加え、新規署名に対する検証情報を含みます。これによって、既存の署名の LTV ステータスが必ず有効に保たれます。

Acrobat で署名済み文書に DSS を追加するには、**「署名」**パネルを開き、**「オプション」**メニュー内で**「検証情報の追加」**をクリックします。

pCOS　DSS の存在は、pCOS パス ***type:/Root/DSS*** を用いてチェックできます。DSS が存在していればその値が 6（***dict***）になっています。DSS そのものがただちに LTV ステータスを保証するわけではないことに留意してください。なぜならそれは、必要な証明書群と失効情報の一部しか含んでいないかもしれないからです。

7.3.4 署名と増分 PDF 更新

デフォルトでは PLOP DS は、電子署名を入力文書に、増分更新として知られている PDF 技法を用いて追加します。増分更新では、入力文書の複製を作成し、署名データをその末

尾に追加することによって、元の文書の内容と構造を温存します。署名オプション *update=false* を用いると、PDF DS は、増分 PDF 更新を追加するのでなく、PDF オブジェクト群のヒエラルキーを書き換えます。更新モードと書き換えモードにおける署名の比較を表 7.5 に挙げます。

表 7.5 更新モードと書き換えモードにおける署名の比較

	更新モード (update=true)	書き換えモード (update=false)
既存の署名群	温存されます	失われます[1]
認証・証明用署名のための DSS を追加	可	可
文書レベルタイムスタンプ・タイムスタンプ付き署名のための DSS（LTV のためには必須）を追加	可	—[2]
既存の DSS を温存	する	する
新しいパラメータを用いて暗号化 (userpassword・masterpassword・permissions)	—	可
最適化	—	可
破損した入力文書のための修復モード	—	可
署名速度	やや速い	やや遅い
以前のバージョン（その署名を適用する前）の文書を Acrobat で復旧できる	可	不可

1. 書き換えモードで既存の署名を用いて文書に署名する場合には、sacrifice={signatures} が必要です。これによって、その署名が除去されることを含意します。この sacrifice オプションが与えられていない場合、署名された入力文書は拒否されます。
2. タイムスタンプのための検証情報を埋め込む必要がある場合には、PLOP DS は常に、DSS を増分更新として追加します。

破損した文書に署名する 更新モードで署名する際には、PDF 相互参照テーブルの中の、あるいは、その文書のオブジェクト構造の中のエラーを、修復することはできません。もし文書が *PLOP_open_document()* において修復を必要とし、その後に更新モードで署名された場合、*PLOP_create_document()* は以下のエラーメッセージを出して失敗します：

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

PLOP_open_document() でオプション *repair=none* を与えれば、破損した文書をその段階で検出できます。その結果、破損した文書に対しては *PLOP_open_document()* が失敗します。修復を要する文書に署名する必要がある場合には、*update=false* を使用する必要があります。その影響については表 7.5 を参照してください。

署名済み文書の旧版へ復帰 増分更新は文書に情報を追加するだけですので、入力文書の構造は温存されます。署名された文書に変更が加えられた場合には、その署名済みバージョンを、その増分更新群を除去することによって再構築することもできます。Acrobat XI/DC でこれを行うには以下のように操作します：

- 署名ページを開き、署名を選択し、プラス記号をクリックすることによってそれを展開します。
- **「署名バージョンを表示」**を選択すると、署名済みバージョンへ復帰します。

署名が別の増分更新の中の DSS を通じて LTV 対応にされている場合には、その更新は、署名済みバージョンへ復帰することによって除去されます。結果として、旧版内の署名は

LTV 対応ですと表示されなくなりますが、文書全体の中ではこの同じ署名が LTV 対応ですと表示されます。これは、増分 PDF 更新を除去した結果であり、その文書全体の中の署名群の実際の LTV ステータスには影響を与えません。この問題はタイムスタンプ署名では起こりません。なぜなら Acrobat は TSA に対しては完全検証を必須としないからです。

この現象は、DSS 内の検証情報が増分更新内に追加されている場合にのみ発生しますので、2 通りの方法で回避できます：

- *dss=false* と設定することにより DSS を避ける。
- *update=false* と設定することにより増分更新を避ける。

どちらの選択肢も、文書レベルタイムスタンプと、埋め込まれたタイムスタンプに対しては効果がありません。なぜならこれらは常に増分更新内に DSS を必要とするからです。

pCOS　増分更新による文書の版の数は pCOS 擬似オブジェクト *revisions* で報告されます。署名はそれぞれが新たな版を生み出しますが、版は他の変更によって生み出されることもあります。たとえば DSS の追加です。ですので、版の数は、その文書内の署名の数よりも大きくなる場合があります。

7.3.5 暗号化を署名と併用

暗号化と署名の併用には注意が必要です。なぜなら、署名をまず行なって、その後に文書を暗号化すると、その署名が無効になってしまうからです。暗号化と署名をシングルパスで行うか、あるいは、暗号化されている入力文書に更新モードで署名することができます。

暗号化と署名をシングルパスで行う　最もシンプルなアプローチは、暗号化と署名の両方をシングルパスで行うことです。入力ファイルが変更される必要がありますので、署名は書き換えモードでのみ可能です。*userpassword*・*masterpassword* といった暗号化オプション群または受信者証明書群を、署名オプション群とともに与えることができます。これらの暗号化パラメータのうちのいずれかが与えられたときには、署名は自動的に書き換えモードで適用されます。すなわち *update* が強制的に *false* になります。

暗号化されている入力文書に更新モードで署名する　暗号化されている入力文書には、更新モードで署名することが可能です。ただしこの場合には、その暗号化パラメータ群は変更できません。ですので以下のようなことになります：

- その入力文書のマスターパスワードを *password* オプションで（あるいは、証明書セキュリティを用いて保護されている文書に対しては、然るべきデジタル ID を）与える必要があります。
- *update=true* を与える場合には、*encryption*・*masterpassword*・*permissions*・*userpassword* オプションは許されず、また、*PLOP_add_recipient( )* を呼び出してはいけません。なぜなら入力文書の値が出力文書のために使われるからです。

7.3.6 証明用署名

証明用（作成者）署名については、「証明用署名」（91 ページ）でさわりを紹介しました。証明用署名を持った文書を開くと、Acrobat は、上端付近の青い文書メッセージバーの中にバッジを表示し、Acrobat の「署名」パネルの中にもその署名を表示します（同じく、それが有効な場合にはバッジとともに）。証明用署名は、その署名を無効にすることなく、その文書に対してどの種類の変更を行うことができるかを指定します（図 7.6・表 7.6 参照）。証明用署名を PLOP DS で作成するには *certification* オプションを用います。

以下の署名オプションは、その署名を無効にすることなくフォーム記入が許される証明用署名を作成します：

表 7.6 証明用署名を無効にすることなく許される文書変更

署名の種類（オプションリスト）	署名を無効にすることなく許される変更				
	フォームフィールドに値を記入	電子署名・ページ追加[1]	注釈を作成・削除・変更	署名フィールドを追加[2]	その他の変更すべて
certification=nochanges	—	—	—	—	—
certification=formfilling	可	可[3]	—	—	—
certification=formsandannotations	可	可[3]	可	—	—
certification=none（すなわち認証署名か文書レベルタイムスタンプ署名）	可	可	可	可	—

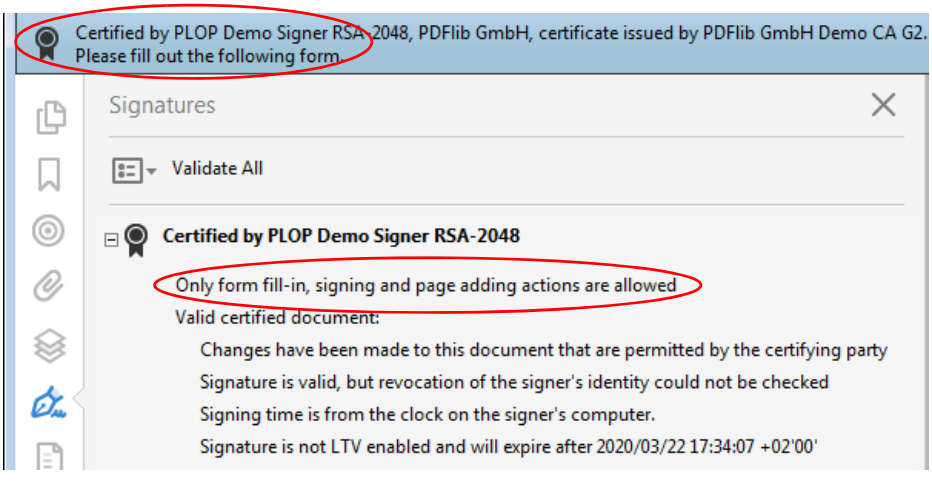
1. ページテンプレートから産み出すという、まれにしか使われない技法を用いてページを追加することが可能かどうかです。「ツール」→「ページ」→「ページの挿入」を用いて手動でページを追加することはできません。
2. 「入力と署名」→「署名を配置」から署名を追加することが可能かどうかです。「ツール」→「フォーム」→「編集」を用いてフォームフィールドを追加することはできません。
3. 署名フィールドをクリックすることによって署名を行うことのみ可能です。Acrobat のメニュー項目から行うことはできません。

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

preventchanges サブオプションを用いると、Acrobat ユーザーインタフェース内の、注釈ツール群等、署名を無効にしてしまうであろうツールを無効にすることができます。こうしておけば、ユーザーが、証明用署名を無効にしてしまうであろう変更を行おうとするおそれはなくなります。Acrobat を用いて文書を証明する際には変更は常に防止されます。この *preventchanges* オプションはデフォルトで *true* に設定されています。*preventchanges=false* の場合、Acrobat はすべての編集ツールを有効にします。しかし、許されない変更は証明用署名をやはり無効にします。

証明用署名は、必ず文書内の最初の署名であるべきですので、すでに署名を含んでいる文書に対しては行なってはいけません。

図 7.6 Acrobat で「フォームフィールドの入力と署名フィールドに署名を許可」とした証明用署名



Acrobat における証明用署名の妥当性　証明用署名が技術的に有効な場合であっても、Acrobat において証明済み文書の利点を完全に活用するには、さらなる必要事項がいくつかあります：

- 証明用署名は、AATL CA からの証明書を用いると（「Acrobat における信頼済みルート証明書」（93 ページ）参照）、最も簡単に作成できます。Adobe Root CA は自動的に必要な信頼設定を持っていますので、構成手順は一切必要ありません。
- PKI からの、Adobe に知られていないルートの下のエンドユーザー証明書で証明用署名を作成しようとする場合には、Acrobat で、必要な信頼レベルをそのルート証明書に割り当てることを推奨します：
  「編集」→「環境設定 ...」→「署名」→「*ID* と信頼済み証明書」→「詳細 ...」→「信頼済み証明書」→そのルート証明書を選択**→「信頼を編集」→「証明済み文書」**を有効に。結果として、この選択されたルートの下の証明書を用いて作成される照明用署名はすべて、有効として受け入れられます。
- 個別の証明書に対して、必要な信頼レベルを設定することもできます。ただし、これは常道ではなく、推奨されません。以下のように操作します：
  「署名」パネルを開き、その証明用署名を選択し、**「証明書を表示 ...」→**証明書チェーン内で署名用証明書（すなわちリストの一番下のもの）を選択し、**「信頼」**タブを開き、**「信頼済み証明書に追加 ...」**をクリックし、通知メッセージダイアログで**「*OK*」**をクリックして、信頼設定を編集します。

上記の方法のいずれも行わないと、Acrobat はその証明用署名に、バッジでなく黄色い三角の印を付け、テキスト**「署名者の証明書は証明済み文書を作成する目的では信頼されていません」**を加えます。

pCOS　証明用署名はpCOSで*signaturefields[...]/sigtype= certification*として報告されます。許されている変更の種類は *signaturefields[...]/permissions* を用いてクエリできます。これはキーワード *nochanges*・*formfilling*・*formsandannotations* のいずれかを返します。

pCOS 擬似オブジェクト *signaturefields[...]/preventchanges* を用いると、禁じられた変更を行うことにより証明用署名がうっかり無効にされてしまわないよう Acrobat のユーザーインタフェース要素が無効にされるかどうかをチェックできます。

変更が禁止されている証明済み文書に署名　証明用署名は、1 つの文書の中で最初の署名である必要がありますので、追加の署名群を最初の署名の後に適用することが通常は可能です。しかし、証明用署名が「変更を許可しない」設定で作成されている場合があります（PLOP DS では *certification=nochanges*）。すると、アプリケーションがこのような入力文書に署名を行いたい場合、そのファイルは署名も含めいかなる変更をも禁じているので、署名を行えばその証明用署名が無効になってしまう、という衝突を引き起こします（承認署名は、更新モードで署名を追加することが可能で、無効になりません）。このような衝突についてはアプリケーション開発者が解決する必要があります。更新モードで署名を行なっても、その証明書はいかなる変更をも許可していないので、同様です。この衝突を解決するには以下の方法がありえます：

- デフォルト動作：既存の証明用署名が新しい署名より優先されますので、入力は拒絶されます。
- 証明用署名を除去し、新たな署名を適用します。これを実現するには *PLOP_create_document( )* のオプション *sacrifice=signatures* を用います。

7.4 証明書失効情報

署名は、その署名用証明書の失効ステータスに関する情報を含むこともできます。この情報は、署名検証ソフトウェアによって、その証明書が署名の時点においていまだ有効であった（失効させられていなかった）ことを保証するために用いられることができます。これを行うには 2 通りの方法があります。

Acrobat XI/DC では、以下のようにして証明書ビューア内で失効情報を確認できます：**「署名」**パネルを開き、その署名を右クリックして、**「署名のプロパティを表示 ...」→「証明書を表示 ...」**を選択し、**「失効」**タブへ行きます（図 7.7・図 7.8 参照）。

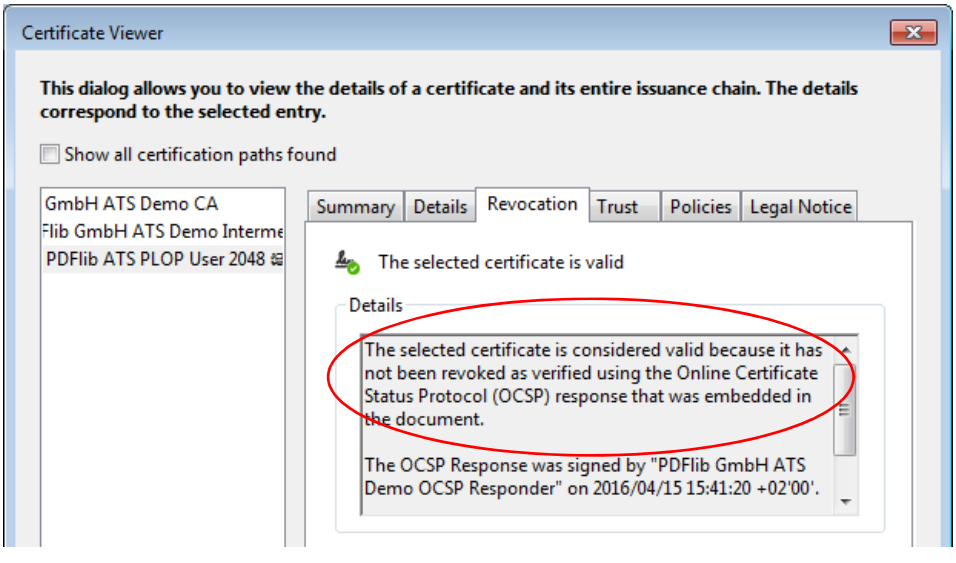
7.4.1 オンライン証明書ステータスプロトコル（OCSP）

注記 `engine=mscapi` **の場合には OCSP 応答の埋め込みには対応していません。**

OCSP の概要　RFC 2560・RFC 6960 に従った OCSP が使用されている場合、署名を行うソフトウェアは、OCSP サーバ（OCSP レスポンダともいいます）へネットワーク要求を送信してその証明書のリアルタイムのステータスをクエリします。OCSP レスポンダは、その CA の発行されたり失効させられたりした証明書群のデータベースへのリアルタイムのアクセスを持つサーバです。この OCSP レスポンダは、そのクエリの時点でその証明書が有効かどうかを確認して、その結果を持った署名済みの応答を返します。この OCSP 応答はその署名に埋め込まれます。

証明書は、RFC 3280 に従った *ocsp* アクセス方式を持った *Authority Info Access*（AIA）という拡張を内容として持っていることもあります。AATL 証明書は通常そのようになっています（「Adobe 認定信頼リスト（Adobe Approved Trust List ＝ AATL）」（93 ページ）を参照）。この拡張は、その証明書を発行した CA に紐付けられた OCSP レスポンダへの URL を内容として持ちます。あるいは、*ocsp* 署名オプションからこの URL を与えることも可能です。PLOP DS が、ある特定の証明書のための OCSP 要求を送信した時には、その OCSP サーバは、その証明書に対するステータス**「有効」・「失効」・「不明」**のいずれかを持った

図 7.7
Acrobat に表示された OCSP 情報



応答を返します。**「有効」**な OCSP 応答を生み出すには、以下の条件がすべて満たされる必要があります：

- ***ocsp*** アクセス方式を持った AIA 拡張がデジタル ID 内に存在するか、あるいは、***ocsp*** 署名オプションの ***source*** サブオプションを与える必要があります。
- そのOCSPレスポンダが、指定されたURLにおいてネットワークを通じて到達可能であり、かつ、***ocsp*** 署名オプションの ***source*** サブオプションの ***timeout*** サブオプションで指定された時間内に応答を送信すること。
- その証明書が、その OCSP レスポンダによって受け持たれている CA によって発行されており、かつ、有効であり（すなわちその有効期限に達していない）、かつ、失効させられていないことを必要条件とする、ステータス**「有効」**を、その OCSP 応答が内容として持っていること。
- その署名日時が、そのOCSP応答の中のエントリ ***thisUpdate*** と ***nextUpdate*** によって定義される期間の中に収まっているか、あるいは（***thisUpdate*** がない場合）、***thisUpdate*** に ***freshness*** サブオプションで指定されている値を加えた日時より後でないこと。どちらのチェックも、ネットワーク遅延や不正確なシステム時刻を補償するために、***maxclockskew*** オプションの値以下のずれは許容します。

OCSP 応答が**「有効」**である場合には、PLOP DS は、その応答を、生成される署名の中へ埋め込みます。そうでない場合には、***critical*** サブオプションに応じて、その使用不能な応答は無視されるか、あるいは署名は生成されません。デフォルトでは PLOP DS は、署名者のデジタル ID の中にもし AIA 拡張があればその中の OCSP レスポンダの URL を用い、有効な OCSP 応答のない状況を黙殺します。しかし、***ocsp*** オプションを用いて OCSP 応答の埋め込みが明示的に要求されている場合には、***critical*** オプションが ***false*** に設定されていない限り、署名を生成するには、**「有効」**な応答が必要です。

OCSP の構成　使用している PKI に応じて、OCSP 応答に関する構成について以下の点を考慮する必要があります：

- 証明書内にAIA拡張がない場合には、***ocsp*** オプションの ***source*** サブオプションを用いて OCSP レスポンダを与える必要があります。
- OCSP　要求を作成するには、署名者の証明書の発行者のための有効な証明書が必要です。これは多くの場合、署名者のデジタル ID の中に含まれています。そうでない場合には別途、***rootcertdir*** ／ ***rootcertfile*** ／ ***certfile*** 署名オプションのいずれかを用いて与える必要があります。
- OCSP レスポンダは、ネットワークコミュニケーションが成功するために認証を必要とする場合がありますので、OCSP　要求ではいくつかの認証オプションに対応しています。
- OCSP の**ノンス**機能は、反射攻撃を防ぐ一方で、キャッシングを妨げるのでパフォーマンスを低下させます。OCSP レスポンダの構成によっては、***nonce*** オプションを用いる必要がある場合があります。以下のようなメッセージを受け取った場合、その OCSP レスポンダはノンス機能に対応していません。この場合には、署名オプション ***nonce=false*** を与えれば、ノンス機能を無効にできます：

  ```
  OCSP response from URL 'http://ocsp.acme.com' for certificate 'CN = PDFlib GmbH...'
  does not contain nonce although it was requested
  ```

  Microsoft OCSP レスポンダは、ノンス処理のために構成されていない場合には、ノンスが要求されると、要求を ***unauthorized*** エラーで拒絶します。この場合には、ノンス機能を無効化するために署名オプション ***nonce=false*** も与える必要があります。

- *ocsp* オプションの *hash* サブオプションを用いると、OCSP 要求・応答の中で証明書を識別するために使用されるハッシュ関数を選択できます。ただし Acrobat XI/DC は、SHA-1 ハッシュ関数を使用した OCSP 応答しか取り扱うことができず、他のハッシュ関数を用いた OCSP 応答を使って署名検証を行うことができません。ですので、*sha1* 以外の値は署名オプション *conformance=extended* を必要とします。

OCSP レスポンダに対する失効確認　OCSP レスポンダの署名用証明書は、OCSP 応答を作成した時点において有効である必要があります。堂々巡り（OCSP レスポンダの証明書はさらなる OCSP 応答を必要とすることになる）を避けるため、OCSP レスポンダの証明書の中には、RFC 2560 に従った *id-pkix-ocsp-nocheck* 拡張を含めることが推奨されています。ほとんどすべての商用 OCSP レスポンダではそのようになっています。あるいは、この証明書は CRL 配布点（*CRLdp*）拡張を内容とすることもできます。

OCSP のオプションリストの例　以下の例では、オプションリストの中の、OCSP 応答の埋め込みに関係する部分のみを示します。他の署名オプション群も適切に加える必要があります。

署名者のデジタル ID の中にある URL を用いて OCSP 応答の埋め込みを試み、*ocsp* アクセス方式を持った AIA 拡張がそのデジタル ID の中で得られない場合にはエラーを発して失敗：

```
ocsp={source={}}
```

または同等表現：

```
ocsp={}
```

AIA 拡張を用いた OCSP 応答の埋め込みが可能であれば行うよう要求し、しかしエラーが出たら黙殺：

```
ocsp={source={} critical=false}
```

または同等表現：

```
ocsp={critical=false}
```

デジタル ID 内に AIA 拡張があっても OCSP 応答を埋め込まない：

```
ocsp=none
```

OCSP レスポンダに対する URL とタイムアウト 1 秒を明示的に与え、たとえデジタル ID 内に AIA 拡張があってもその中のエントリをオーバライド：

```
ocsp={source={url={http://ocsp.acme.com/} timeout=1000} }
```

OCSP 試行が成功しなかった場合にはその署名処理を絶対に行わないようにし、かつ、ノンス機能に対応していない OCSP レスポンダに対してはノンス機能を無効に：

```
ocsp={critical nonce=false}
```

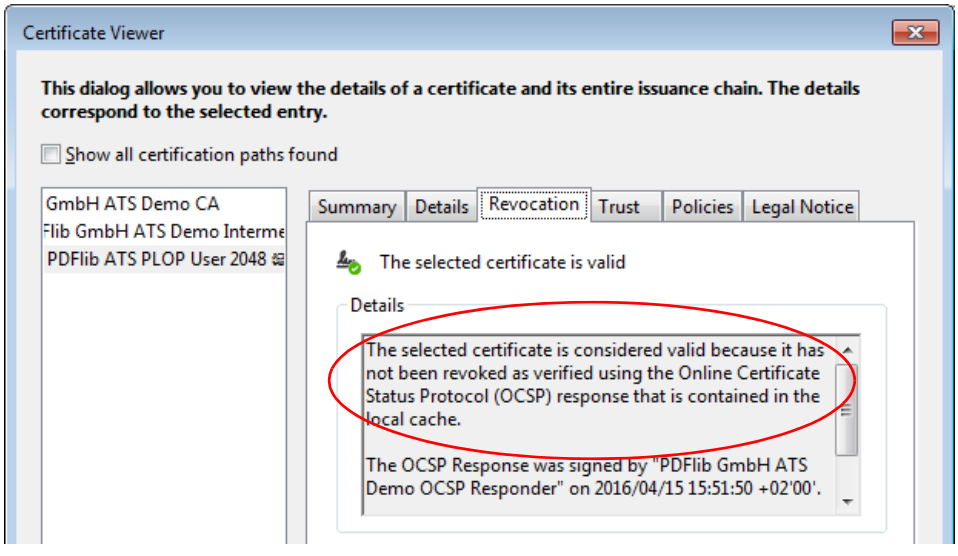
7.4.2 証明書失効リスト（CRL）

注記 `engine=mscapi` **の場合には CRL の埋め込みには対応していません。**

CRL の概要　RFC 3280 およびその後継 RFC 5280 に従った CRL が用いられている場合には、CA は定期的に（1 日 1 回等）、まだ期限が切れてはいないけれども失効させられている証明書群の署名済みリストを作成します。このリストは、署名ソフトウェアからの入手を可能にされ、そして署名内へ埋め込まれます。このリストは、ネットワークを通じて取得することもできますし、ローカルに保管することもできます。CRL は、ある特定の継続期間（1 日等）を持っており、その継続期間が尽きる前にリフレッシュされる必要があります。CRL は、任意の数の失効させられた証明書を取り扱うことができますので、通常、OCSP 応答よりはるかに大きく（数メガバイトにも）なり、しかもそのサイズは事前にはわかりません。CRL はまるごと PDF 出力内へ埋め込まれますので、この種の失効情報は署名済み PDF 文書を肥大化させます。PLOP DS は CRL をいくつかの取得源から取得できます：

- 証明書は、***CRL 配布点***（***CRLdp***）という拡張を内容として持っていることがあります。AATL 証明書は必ずそのようになっています（「Adobe 認定信頼リスト（Adobe Approved Trust List ＝ AATL）」（93 ページ）を参照）。この拡張は、1 個ないし複数の CRL リソースのネットワーク URL を内容としています。PLOP DS は、CRL を取得できるまで、この ***CRLdp*** 拡張の中のすべてのエントリを試します。使用可能な CRL が見つかった場合には、それは署名内へ、あるいは文書セキュリティストア（Document Security Store ＝ DSS）内へ埋め込まれます（7.3.3 節「文書セキュリティストア（DSS）」（110 ページ）参照）。この ***CRLdp*** 拡張は、CRL を必要とする証明書それぞれについて、OCSP 応答の入手可能性と、照応する ***critical*** オプションに応じて、評価されます。
- この ***CRLdp*** 拡張のかわりに、署名用証明書のための CRL の取得を、***crl*** オプションを用いて構成することもできます。そのサブオプション ***source*** は、CRL が動的に取得されるネットワークアドレスを指し示します。サブオプション ***filename*** は、DER エンコーディングの静的なローカル CRL ファイルを指し示します。
- 署名用証明書とその他すべての関与する証明書のための 1 個ないし複数のローカル CRL ファイルを、***crldir/crlfile*** 署名オプションを用いて PEM エンコーディングで与えることもできます。

図 7.8
Acrobat で表示された CRL 情報



署名者の証明書が CRL に含まれている場合、それはその発行した CA によって失効させられており、すなわち、もはやそれを使用して有効な署名を作成することはできなくなっています。この場合、*PLOP_prepare_signature()* は以下のようなエラーメッセージを発して失敗します：

```
Certificate verification failure for certificate with subject 'C = DE, L = Munich, O =
PDFlib GmbH, CN = PLOP Demo Signer RSA-2048': certificate revoked
```

PLOP DS は、CRL を、それが期限切れになるまで使用します。ある特定の CRL が、その継続期間が尽きたためにもう使えなくなった時にのみ、PLOP DS は新しい CRL をサーバからダウンロードします。

CRL のオプションリストの例　以下の例では、オプションリストの中の、CRL の埋め込みに関係する部分のみを示します。他の署名オプション群も適切に加える必要があります。

署名者のデジタル ID の中にある URL を用いて CRL の埋め込みを試み、*CRLdp* 拡張がそのデジタル ID の中で得られない場合にはエラーを発して失敗：

```
crl={source={}}
```

または同等表現：

```
crl={}
```

CRLdp 拡張を用いた CRL の埋め込みが可能であれば行うよう要求し、しかしエラーが出たら黙殺：

```
crl={source={} critical=false}
```

または同等表現：

```
crl={critical=false}
```

デジタル ID 内に *CRLdp* 拡張があっても署名用証明書やその他任意の証明書に対して CRL を取得しようと試みない。これは、署名を行うコンピュータがオフラインである場合等、オンライン取得が必ず失敗することがわかりきっている場合には合理的です：

```
crl=none
```

CRL サーバに対する URL とタイムアウト 1 秒を明示的に与え、たとえデジタル ID 内に *CRLdp* 拡張があってもその中のエントリをオーバライド：

```
crl={source={url={http://crl.acme.com/} timeout=1000} }
```

ローカルディスクファイルの内容である CRL を与える：

```
crlfile={certs.pem}
```

7.4.3 OCSP か CRL か

失効情報を含める方式として最も然るべきものを選ぶにあたっては、以下の要素が意味を持ちます：

- OCSP はリアルタイムは証明書ステータス情報を提供します。1 つの OCSP 応答はただ 1 つの証明書のみを取り扱いますので、そのサイズはわずか数キロバイトであると予測

できます。他方で、OCSP は必ず OCSP レスポンダへのネットワーク接続を必要とします。

- CRL が OCSP に勝る利点は、ローカルに保管できるのでネットワークオーバヘッドを避けることができる点です。難点は、ローカルに保管された CRL は、頻繁にリニューアル（すなわち発行・ダウンロード）されない限り、内容が古くなってしまうおそれがある点です。
- CA の証明書を失効させる必要が生じることはまれですので、CA 群に対する CRL は通常、エンドユーザー証明書に対する CRL よりもはるかに小さいです。
- 同様に、HSM が破壊されたり盗まれたりすることはまれですので、HSM ベースの証明書群に対する CRL は多くの場合、非常に小さいです（その CRL が、HSM ベースの証明書群のみを対象としており、ファイルベースの証明書群を対象としていないことを前提として）。
- 法令や、署名に関する内規によって、両方式のうちのいずれかが義務付けられている、あるいは禁止されている場合がありえます。

デフォルトでは PLOP DS は、有効な OCSP 応答が得られない場合にのみ CRL を署名内へ埋め込みますが、この動作は、***ocsp・crl*** オプションと ***critical*** サブオプションを用いて変えることもできます。

失効情報が必ず埋め込まれるようにするには以下のオプションリストを用います。この場合、OCSP が有効な応答を与えない場合にのみ CRL が取得されます：

```
ocsp={critical=false source={url={http://ocsp.acme.com/}}} ↵
 crl={critical=true  source={url={ http://crl.acme.com/}}}
```

この ***ocsp・crl*** オプションが失効情報の埋め込みを制御するのは、署名用証明書に対してのみであり、CA または TSA 証明書群が関与していてもそれらに対しては制御しないことに留意してください。

7.5 タイムスタンプ

7.5.1 タイムスタンプの構成

電子署名は、信頼された時刻サーバから取得された日時情報を含むこともできます。このようなサーバを、時刻認証局（Time-Stamp Authority ＝ TSA）ともいいます。署名を行うコンピュータから採られた時刻（容易にごまかしが可能）とは異なり、信頼されたサーバから取得されたタイムスタンプは、署名の時点について、署名済みで信頼に足る情報源を提供します。PLOP DS は、RFC 3161・RFC 5816・ETSI EN 319 422 に従ったタイムスタンプ処理に対応しています。このタイムスタンプ処理要求は、その生成される署名のハッシュを含んでいますので、そのタイムスタンプは、その署名が特定の時点に作成されていることを確証します。このタイムスタンプは、その生成される PDF 署名の中へ埋め込まれます。

PLOP DS は、それぞれのタイムスタンプを、生成された文書の中へそれを埋め込む前に検証します。要求されたタイムスタンプが検証できない場合にはエラーが発生します。たとえば要求された認証情報が構成されていない場合などです。適格なタイムスタンプを生成する TSA など、然るべき TSA を選択して構成するのはユーザー側の責務です。

選択した TSA に応じ、タイムスタンプを作成するための構成について、以下の点を考慮する必要があります：

- 最も重要な情報は、TSA へ到達できるネットワークアドレスです。これを与えるには ***source*** サブオプションの ***url*** サブオプションを用います。あるいは署名者のデジタル ID から採ることも可能です（「デジタル ID 内のタイムスタンプ拡張」（122 ページ）参照）。
- TSAを信頼するには、そのTSA証明書を発行したCAが信頼されている必要があります。この TSA の CA 証明書は、署名を検証する際、他の CA 証明書群と同様に処理される必要があります。詳しくは「すべてのチェーンに対して信頼済みルート証明書を構成」（128 ページ）を参照してください。これは特に LTV 対応署名を作成する際に重要です。AATL ヒエラルキー群（「Adobe 認定信頼リスト（Adobe Approved Trust List ＝ AATL）」（93 ページ）参照）のいずれかの下にある TSA を利用する場合には、その TSA 証明書の発行者、ないし発行者群のチェーンは、Acrobat に信頼済みルートとして知られています。ただし、その TSA CA 証明書を ***certfile*** オプションで PLOP DS に与える必要がある場合があります。
- TSA は、クライアントがタイムスタンプ要求を作成する際に、ある特定のハッシュアルゴリズムを使用するよう要求する場合があります。デフォルトでは PLOP DS は SHA-256 アルゴリズムを使用しており、これは現在のすべての TSA で動作します。これ以外のハッシュ関数を与えるには ***hash*** サブオプションを用います。なお、タイムスタンプ署名内で使用されるハッシュアルゴリズムを指定することはできません。なぜならそれは完全にその TSA の制御下であるからです。
- TSA のなかには、自由にアクセスできるものもありますが、一部の商用 TSA は、アクセスを制限するためにユーザー名とパスワードを要求します。権限のないアクセスには以下のようなメッセージが返されます：

  ```
  Network response from URL 'https://timestamp.acme.com/tsa' has bad status code 401
  ('Unauthorized')
  ```

  権限パラメータ群を与えるには、URL に含めるか、あるいは ***source*** ネットワークサブオプションの ***username/password*** サブオプションを用います。
- TSA が SSL アクセス（すなわち ***https***）を要求する場合には、そのサーバの SSL ルート証明書を、***sslcertdir/sslcertfile*** オプションを用いて与える必要があります。そうしないと以下のようなメッセージを返されてしまいます：

```
Document timestamp request to 'https://timestamp.acme.com/tsa' failed
('Peer certificate cannot be authenticated with given CA certificates')
```

必要なサーバ証明書を与えるのではなく、オプション *sslverifypeer=false* を用いることによって、サーバ証明書の確認をスキップすることもできます。ただし、そのセキュリティ上の影響を認識している場合に限ります。

- TSA のなかには、ポリシー OID（オブジェクト識別子）を要求するものもあり、これを与えるには *policy* サブオプションを用います。この OID の適切な値についてはその TSA とすり合わせておく必要があります。このポリシー OID は、Acrobat の**「署名のプロパティ」**ダイアログ→**「証明書を表示 ...」**に表示されます。

7.5.2 タイムスタンプ付き署名

注記 `engine=mscapi` **の場合にはタイムスタンプ付き署名には対応していません。**

認証・証明用署名は、埋め込まれたタイムスタンプを内容として持つこともできます。Acrobat 7 およびそれ以降がタイムスタンプ付き署名に対応しています。

デジタル ID 内のタイムスタンプ拡張　デジタル ID は、時刻認証局の URL を内容とする *TimeStamp* 拡張を内容として持っていることがあります。これにより、TSA の詳細を与える必要なく、タイムスタンプを埋め込んだ署名を行うことが可能になります。この *TimeStamp* 拡張は通常、AATL（***Adobe* 認定信頼リスト**）プロバイダ群によって発行される証明書（「Adobe 認定信頼リスト（Adobe Approved Trust List ＝ AATL）」（93 ページ）を参照）の中には含まれています。一部の AATL プロバイダはタイムスタンプを数量限定で無償で提供しています。

この *TimeStamp* 拡張があり、かつ認証を要求しない URL を内容としている場合には、PLOP DS はタイムスタンプを作成するために、その指定された TSA へアクセスを試みます。この場合には、タイムスタンプを作成するために *url* サブオプションを与える必要はありません。しかし、認証を要求する TSA を使用するには、たとえ *TimeStamp* 拡張内でその TSA が指定されていても、その TSA の完全な詳細を明示的にオプションリストで指定する必要があります（後述の例を参照）

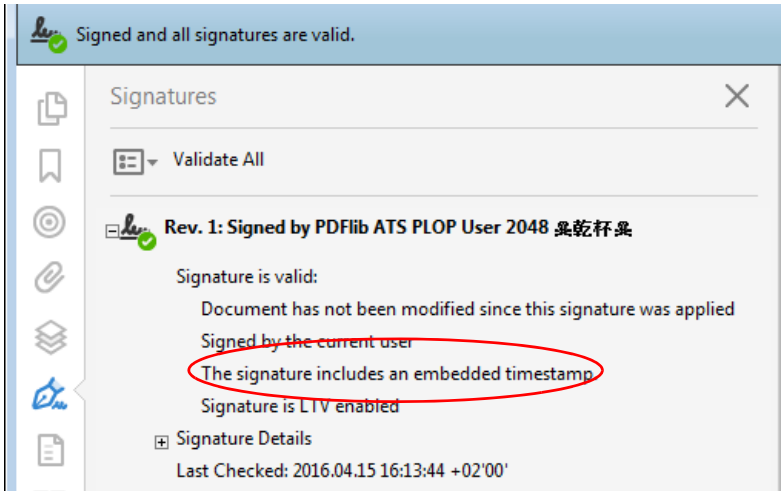


図 7.9
Acrobat におけるタイムスタンプ付き署名

タイムスタンプのオプションリストの例　以下の例では、オプションリストの中の、タイムスタンプの埋め込みに関係する部分のみを示します。他の署名オプション群も適切に加える必要があります。

指定した URL にある TSA から取得したタイムスタンプを用いて、デフォルトハッシュアルゴリズム SHA-256 を使用して署名にスタンプ：

```
timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

タイムスタンプを取得するには TSA がユーザー名とパスワードを要求する所のタイムスタンプを用いて署名にスタンプ：

```
timestamp={source={url={http://timestamp.acme.com/tsa} username=demo password=demo}}
```

TSA がダイジェスト認証を要求する所のタイムスタンプを用いて署名にスタンプ

```
timestamp={source={url={http://timestamp.acme.com/tsa} httpauthentication=digest ↵
        username=demo password=demo }}
```

TSA が SSL を通じてアクセスされる必要がある場合には、そのサーバの SSL 証明書を、オプション *sslcertdir/sslcertfile* を用いて与える必要があります。そのサーバの SSL 証明書が手に入らない場合には、オプションを用いてサーバ認証をスキップすることも可能です。ただし、そうすることのセキュリティ上の意味を認識している場合に限ります：

```
timestamp={source={url={https://timestamp.acme.com/tsa}} sslverifypeer=false}
```

署名者のデジタルIDの中にあるURLを用いてタイムスタンプを署名へ埋め込もうと試み、そのデジタル ID の中で *TimeStamp* 拡張が得られなければエラーを発して失敗：

```
timestamp={source={}}
```

または同等表現：

```
timestamp={}
```

たとえデジタル ID 内に *TimeStamp* 拡張があってもタイムスタンプを埋め込まない：

```
timestamp=none
```

7.5.3 文書レベルタイムスタンプ署名

文書レベルタイムスタンプは、PAdES パート 4 で導入されており、ISO 32000-2 に盛り込まています。

タイムスタンプ付き署名と文書レベルタイムスタンプの違い　タイムスタンプ付き署名と同様に、文書レベルタイムスタンプは、ある特定の時点に紐付いたステータス情報を提供します。ただし、前者ではタイムスタンプは主たる署名の属性であるのに対して、文書レベルタイムスタンプは有効な署名そのものです。それはデジタル ID を必要としません。なぜなら署名する人や主体というものがないからです。かわりに文書レベルタイムスタンプは、時刻認証局（TSA）へのネットワーク要求を通じて作成されます。文書レベルタイムスタンプは、ある特定の文書が、そのタイムスタンプで示された時刻に存在していたことを保証します。

注記 `engine=mscapi` **の場合には文書レベルタイムスタンプ署名には対応していません。**

文書レベルタイムスタンプのオプションリストの例　以下の例では、文書タイムスタンプを作成するための完全な署名オプションリストを示します。署名用証明書が必要ありませんので、他の署名オプションは一切必要ありません。

指定したURLにあるTSAから取得した文書レベルタイムスタンプを、デフォルトハッシュアルゴリズムSHA-256を使用して追加：

```
doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

ユーザー名とパスワードを要求するTSAからの文書レベルタイムスタンプを追加：

```
doctimestamp={source={url={http://timestamp.acme.com/tsa}} username=demo password=demo}
```

ダイジェスト認証を要求するTSAからの文書レベルタイムスタンプを追加：

```
doctimestamp={source={url={http://timestamp.acme.com/tsa} httpauthentication=digest ↵
        username=demo password=demo}}
```

pCOS　文書レベルタイムスタンプはpCOSで *signaturefields[...]/sigtype=doctimestamp* として報告されます。

7.5.4 トラブルシューティングと対応していないTSA種別

タイムスタンプ応答のサイズ超過　PLOP DSはタイムスタンプのサイズをあらかじめ知る必要があります。PLOP DSは、TSAから受け取るタイムスタンプ応答の最大サイズについて内蔵の値を使用します。タイムスタンプ応答がこの最大値を超えている場合には以下のエラーが発生します：

```
Not enough space reserved for signature contents (reserved XXX bytes, need YYY bytes)
```

この場合には、署名オプション *timestampsize* を用いてこの最大値を大きくすることが可能です。この *timestampsize* の内部的デフォルト値は表8.7(155ページ)に記されています。

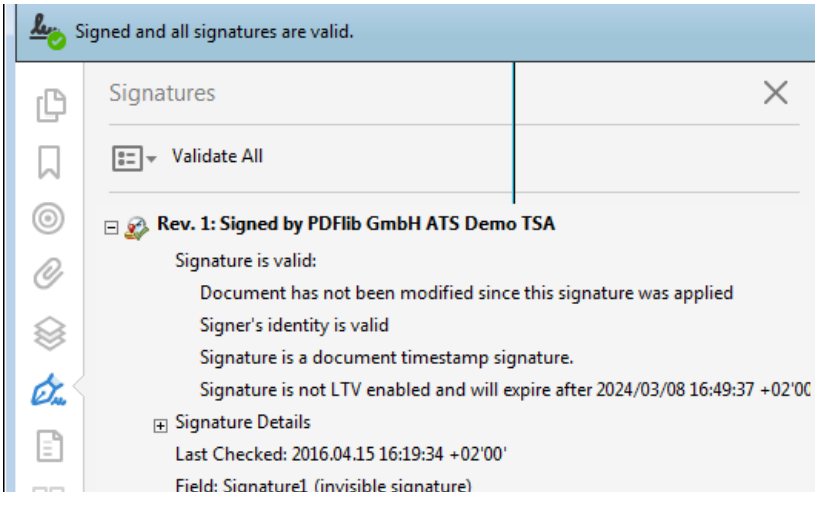


図 7.10
Acrobatにおける文書レベルタイムスタンプ

以下に述べる状況では、TSA を使用して PLOP DS で PDF 文書に署名を行うことはできません。

属性証明書　PLOP DS は属性証明書には対応していません。TSA がそれを使用している場合、PLOP DS は以下のエラーメッセージを発します：

```
Timestamp authority 'http://adobe-tsa.entrust.net/TSS/HttpTspServer'
uses unsupported protocol ('wrong tag')
```

属性証明書はとりわけ、TSA の時刻監査証明書（Time Auditing Certificate ＝ TAC）のために用いられます。TSA 製品のなかには、RFC 2630 に従った新しい CMS 文法を用いて TAC を符号化するものがあります。PLOP DS はこれに対応していません。ただし、RFC 3126 に従った署名済み属性の中に TAC を入れる等の代替方式を用いて TAC を符号化するようそのような製品を構成することも可能です。

鍵使用方法拡張内に「クリティカル」フラグがない　タイムスタンププロトコル RFC 3161 では、TSA 証明書が**「拡張鍵使用方法」**拡張を含み、その値が**「タイムスタンプ」**で、かつこの拡張がクリティカルとして標識されていることを必須としています。この拡張が TSA 証明書内にありながら、クリティカルとして標識されていない場合には、Acrobat はその署名を無効として拒絶します。

PLOP DS は、このような TSA 証明書を使用して生成されたタイムスタンプを、以下のエラーメッセージとともに拒絶します：

```
Signature verification of timestamp failed: certificate verify error:
Verify error:unsupported certificate purpose
```

「拡張鍵使用方法」フィールドに「クリティカル」フラグが設定されていない TSA 証明書を使用して Acrobat で文書タイムスタンプを作成しようとすると、以下のエラーメッセージが出ます：

```
Error encountered while signing:
Certificate is not valid for the usage
```

そのような TSA を使用してタイムスタンプを埋め込んだ証明用または認証署名を Acrobat で作成すると、成功しますが、検証の際には、生成された署名の中のタイムスタンプが以下のメッセージとともに拒絶されます：

```
The signature includes an embedded timestamp but it is invalid
```

Authenticode タイムスタンプサーバ　Authenticode は、Microsoft のタイムスタンププロトコルであり、コード署名をその主用途としています。Authenticode は、RFC 3161 でなく、古い RFC 2985/PKCS#9 に基づいていますので、PDF も PLOP DS もこれには対応していません。

PLOP DS は、Authenticode TSA を使用して生成されたタイムスタンプを、以下のようなエラーメッセージとともに拒絶します。

```
Unexpected content type 'text/html;charset=ISO-8859-1' in reply to timestamp request to
URL 'http://timestamp.entrust.net/TSS/AuthenticodeTS'
(expected content type 'application/timestamp-reply')
```

または

```
Unexpected content type 'application/timestamp-query' in reply to timestamp request to
URL 'http://timestamp.verisign.com/scripts/timstamp.dll'
(expected content type 'application/timestamp-reply')
```

Authenticode TSA を Acrobat で使用しようとすると以下のエラーメッセージが出ます：

```
Error encountered while signing:
Error encountered while BER decoding
```

7.6 長期検証（LTV）

7.6.1 LTV の概念と Acrobat の対応

長期検証（Long-Term Validation ＝ LTV）は、署名を、その署名用証明書が期限切れになったり失効させられたりした後でもなお検証できるという意味を持ちます。これは、署名済み文書を長期間にわたってアーカイブするためには重要な特徴です。この LTV の概念は、PAdES パート 4（ETSI TS 102 778-4）で論じられており、Acrobat XI/DC はこれに対応しています。LTV 署名は eIDAS 規則の要件に準拠しています。

署名を LTV 対応にするには、その完全な証明書チェーンと、関与するすべての証明書に対する失効情報、すなわちまとめて検証情報と呼ばれるものを、その署名の中へ、あるいは DSS（7.3.3 節「文書セキュリティストア（DSS）」（110 ページ）参照）内へ埋め込む必要があります。LTV のために署名関連データを追加で埋め込む必要があることから、その署名済み文書は概して、LTV 対応でない署名よりも大きくなります。

注記 `engine=mscapi` **の場合には、LTV 対応署名には対応していません。**

LTV 対応署名は、埋め込まれたタイムスタンプを含むべきですが、これは厳格な要請ではありません。LTV 対応署名の継続期間を、その関与する証明書群のうちのいずれかが期限切れになるか失効させられる前に文書レベルタイムスタンプ署名を追加することによって延ばすことも可能です。

LTV ステータスは、絶対的に定義されるのではなく、信頼済みルート証明書群の 1 つの集合との関連において定義されます。構成によって、ある特定の署名が、ある構成では LTV 対応と見なされ、別の構成では LTV 対応でないと見なされることもありえます。たとえば、PLOP DS 内と Acrobat 内とで別々の信頼済みルート群を構成すれば、LTV ステータスは異なる可能性があります。

Acrobat における LTV ステータス　Acrobat XI/DC は**「署名」**パネル内に、ステータス行「署名は LTV 対応です」または「署名は LTV 対応ではなく、… を過ぎると有効期限が切れます」を表示します（図 7.11 参照）。LTV ステータスに関して以下のことに留意してください：

- 関与するすべての証明書に対するルート証明書（群）が Acrobat 内で信頼済みとして構成されている必要があります（「Acrobat における信頼済みルート証明書」（93 ページ）参照）。
- Acrobat では、任意の有効な署名について、その直接の署名用 CA 証明書を信頼済みルートストアへ追加することによって、LTV 対応ですと強制的に表示させることができます。このことは、その構成を考えに入れていないとその LTV ステータスについて混乱を招きかねません。これはまた、自己署名証明書を使用して作成された証明書が信頼済みルート証明書に追加されればLTV 対応として扱われることにもつながります。
- Acrobat XI/DC は、埋め込みタイムスタンプがなくても LTV ステータスを認めます。タイムスタンプが埋め込まれている場合、Acrobat はその TSA 証明書に対して検証情報を要求しません。PLOP DS はもっと厳格であり、かつ、TSA 証明書に対して完全な検証情報の要求も行います。Acrobat は、TSA 証明書に対する埋め込み OCSP 応答を、それが検証日時のわずか数分前に作成されたのでない限り、使用しません。
- Acrobat XI/DC は、OCSP 応答について、SHA-1 ハッシュ関数にのみ対応しています（「OCSP の構成」（116 ページ）参照）。ですので、それ以外のハッシュ関数が使用されている場合、完全な検証情報が実際には利用可能でありながら、Acrobat はその LTV ステータスを正しく表示しないことがあります。

- Acrobat で以下の設定を有効にすると、LTV ステータスが表示されなくなりますので、有効にしてはいけません：**「環境設定」→「署名」→「検証」→「詳細 ...」→「検証時刻」→「署名の検証に使用する時刻 ：現在の時刻」**。
- LTV ステータスは、以前の版へ復帰することによって失われる場合があります。詳しくは「署名済み文書の旧版へ復帰」（111 ページ）を参照してください。

7.6.2 PLOP DS を用いた LTV 対応署名

以下のオプションを与えると、PLOP DS は、すべての検証情報が得られるならば、LTV 対応署名を作成します。そうでなければ署名を作成せずエラーを発します：

```
ltv=full
```

このオプション単独では、必ず LTV 対応署名になるわけではなく、すべての必要事項が満たされていることを確認するだけです。証明書が見つからない場合、または検証情報を取得できなかった場合には、PLOP DS はエラーメッセージを発します。ですので、すべてのエラーメッセージを徹底的に分析することが重要です。

デフォルト設定 *ltv=try* の場合には、入手可能なすべての失効情報が、署名される文書の中へ埋め込まれますが、たとえその検証情報が LTV ステータスの実現には不充分であってもその署名呼び出しは失敗しません。

すべてのチェーンに対して信頼済みルート証明書を構成　関与しているすべての証明書を完全に検証するために、PLOP DS はすべての証明書に対してトラストアンカーを必要とします。その正確な数は PKI 構成によります。信頼済みルート証明書群を、*rootcertdir* または *rootcertfile* オプションを用いて与える必要があります。これは最低限、署名用証明書のためのチェーンの最上位にあるルート CA の証明書については行う必要があります。関与するすべての証明書チェーンの最上位にただ 1 つのルート CA があるのでない限り、その他に TSA 用等のルート証明書群も必要になる場合があります。

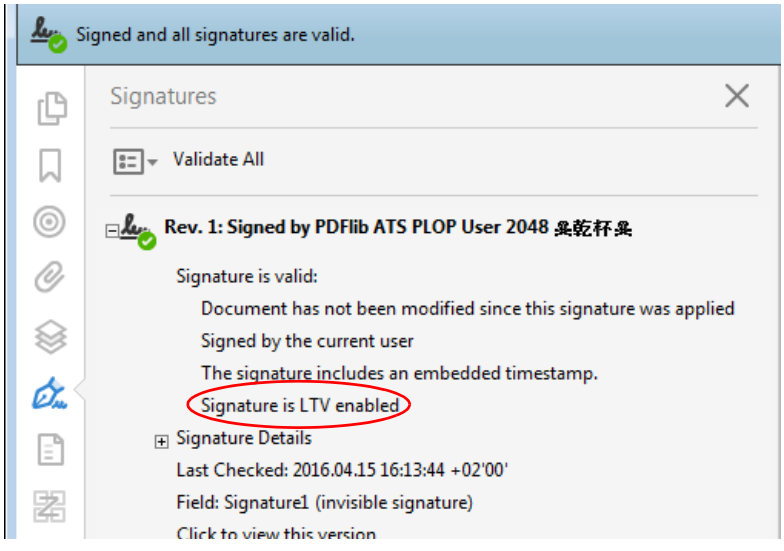


図 7.11
Acrobat における LTV 対応署名

中間 CA 証明書（群）を構成　残りの証明書チェーン（すなわち、ルート証明書と署名用証明書等関与している証明書との間のすべての中間 CA）が、PLOP DS がそれを署名内へ埋め込めるよう、入手可能である必要があります。署名用証明書や、その他、OCSP レスポンダ証明書や TSA 証明書等関与している証明書に対する CA 証明書群は、以下の場所で検索されます：

- CA 証明書群を、*certfile* オプションを用いて与えることもできます。
- （*engine=mscapi* の場合には不可）署名用証明書に対する CA 証明書群を、その署名者のデジタル ID を内容として持つ PKCS#12 ファイルの中に含めることもできます。
- （*engine=mscapi* の場合のみ）CA 証明書群は、Windows 証明書ストア内で検索されます。
- 証明書は、RFC 3280 に従った *calssuers*（認証局発行者）アクセス方式を持った *Authority Info Access*（AIA）拡張を内容として持っていることがあります。この拡張は、署名用証明書を発行した CA の証明書と、場合によっては中間 CA 証明書群を発行した CA の証明書をダウンロードできる、1 個ないし複数の URL を内容としています。プロトコル *http・https・ftp* に対応しています。

証明書はその AIA 拡張内で LDAP プロトコルを指定している場合がありますが、現在のところ PLOP DS はそれには対応していません。この場合には、LDAP ブラウザ[1] を使って手作業で LDAP を通じてその CA 証明書を取得すれば、それを上述のオプション群に与えることができます。これはそれぞれの署名用証明書について 1 回だけ行えば足ります。

どの CA 証明書群を構成する必要があるか　LTV ステータスを実現するための具体的な必要事項は PKI 構成によって異なります。多くの場合は以下の手順で充分です：

- 多くの商用 CA によって発行された証明書は *calssuers* アクセス方式を持った AIA 拡張を含んでいます。ですので PLOP DS は、その署名用証明書に対する CA 証明書群のチェーンを自動的にダウンロードできます。そのルート CA 証明書だけを、*rootcertdir* または *rootcertfile* オプションを用いて与える必要があります。
  calssuers アクセス方式を持った AIA 拡張が署名用証明書内にないときは、多くの場合、必要なルート証明書（群）をその CA のウェブサイトからダウンロードできます。
- TSA と OCSP レスポンダの証明書は自動的に取得されます。これらの証明書、ないしはいずれかの中間証明書が署名用証明書とは異なるルート CA によって発行されている場合には、そのルート証明書を、*rootcertdir* または *rootcertfile* オプションを用いて与える必要があります。
- CRL は多くの場合、クエリされている証明書を発行したのと同じ CA によって署名されています。しかし、もしも CRL が別の CA によって署名されている場合には、その照応する CA 証明書を、*certfile* オプションを用いて与える必要があります。なぜならそれを自動的にダウンロードすることはできないからです。CRL を署名するために使用された証明書が、署名用証明書とは異なるルート CA によって発行されている場合（別の PKI に基づく TSA に対する CRL 等）には、そのルート証明書を、*rootcertdir* または *rootcertdir* オプションを用いて与える必要があります。

validate=full または *ltv=full* の場合、PLOP DS は、必要な CA 証明書が見つからないとき、エラー「*unable to get local issuer certificate*」を発します。この場合には、その足りない証明書を、オプション *rootcertdir・rootcertfile・certfile* のいずれかで与える必要があります。以下のメッセージ：

```
Certificate verification failure for certificate with subject '...':
self signed certificate in certificate chain
```

1. たとえば www.ldapbrowser.com/ で入手可能なフリーの Softerra LDAP Browser。

は通常、信頼済み自己署名証明書を、*rootcertfile* または *rootcertdir* オプションでなく *certfile* オプションで与えたときに発せられます。

署名用証明書に対する失効情報　署名用証明書に対する失効情報を、以下のいずれかの手段によって与える必要があります：

- 署名者の証明書の中の *AIA* 拡張か *ocsp* オプションを通じた OCSP。
- 署名者の証明書の中の *CRLdp* 拡張か *crl* オプションを通じた CRL。既存の CRL 群を、*crldir*・*crlfile* オプションを用いて与えることもできます。

ocsp・*crl* オプションの *critical* サブオプションを用いると、必ずその署名用証明書に対する OCSP または CRL 情報の取得が成功した場合にのみ署名が作成されるようにすることができます。詳しくは 7.4 節「証明書失効情報」（115 ページ）を参照してください。

その他の関与している証明書群に対する失効情報　証明書チェーン内のすべての CA の証明書に対する失効情報と、CRL と OCSP 応答に署名するために使用されているすべての CA の証明書に対する失効情報も、入手可能である必要があります。ただし例外として以下の場合には失効情報は必要ありません：

- *rootcertdir* または *rootcertfile* オプションへ与えたルート CA 証明書群。
- *id-pkix-ocsp-nocheck* 拡張を含んでいる（通常は含んでいます）、OCSP レスポンダの証明書。

署名用証明書以外の証明書群に対する失効情報を、以下のいずれかの手段によって与えることができます：

- 証明書内の *AIA* 拡張を通じた OCSP。
- 証明書内の *CRLdp* 拡張を通じた CRL。既存の CRL 群を、*crldir*・*crlfile* オプションを用いて与えることもできます。

LTV のオプションリストの例　1 番目の例では、PKI が以下のように設定されている場合を考えます：

- 署名者のデジタル ID が、CA 証明書群のチェーンを、その PKCS#12 ファイル内に内容として持っているか、あるいは、ルート証明書以外の各証明書が、*calssuers* アクセス方式を持った AIA 拡張を内容として持っている。
- 署名者のデジタル ID と、ルート証明書以外のチェーン内のすべての CA 証明書が、*ocsp* アクセス方式を持った AIA 拡張か、*CRLdp* 拡張を内容として持っている。
- OCSP レスポンダの証明書が *id-pkix-ocsp-nocheck* 拡張を内容として持っている。

この場合には、LTV ステータスを実現するには *rootcertfile* オプションだけが（そのデジタル ID に対するオプション群に加えて）必要です。オプション *ltv=full* を用いると、LTV の必要事項への違反が必ず検出されるようになり、LTV ステータスが実現できない場合には署名が作成されなくなります：

```
digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=root1.pem
```

タイムスタンプを埋め込むためには、LTV ステータスを実現するには、その TSA 証明書に対する失効情報も入手可能である必要があります。その TSA 証明書が、*ocsp* アクセスメソッドを持った AIA 拡張か、*CRLdp* 拡張を内容として持っており、かつ、そのルート CA が署名用証明書と同じであることが理想です。この場合には、LTV ステータスを実現するためにさらなるオプション指定は必要ありません：

```
digitalid={filename=demo_signer_rsa_2048.p12} passwordfile=pw.txt ltv=full ↵
        rootcertfile=root1.pem timestamp={source={url={http://timestamp.acme.com/tsa}}}
```

しかし、そのTSAが別のルートCAに基づいている場合には、そのTSAルートを*rootcertfile*オプションで与える必要があります（ファイル *root1+2.pem* が、必要なルート証明書をPEMエンコーディングで両方とも内容としていると前提）：

```
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 各種 CAdES・PAdES 署名規格

7.7.1 各種 CMS・CAdES 署名

欧州電気通信標準化機構（ETSI）[1] は、EU 加盟諸国間の電子署名の調和を図るため、数多くの電子署名規格を発行しました。ETSI 規格群は世界のそれ以外の地域でも非常に影響力があります。これらは PDF 2.0 標準 ISO 32000-2 内で参照されており、さまざまな RFC にも取り入れられています。

CMS・CAdES 署名　長い間、PDF 署名は CMS（Cryptographic Message Syntax ＝暗号メッセージ構文）に基づいていました。これは、RFC 5652 で仕様化されており、さまざまなインターネットプロトコルで広く使われています。PDF 内の CMS 署名は、署名辞書内において、サブフィルタ ***adbe.pkcs7.detached*** か、あるいはもっと古い何らかの非推奨のエントリを使用しています。CMS 署名は、すべてのバージョンの Acrobat で作成・検証できます。

CAdES（***CMS Advanced Electronic Signatures*** ＝ CMS 高度電子署名）は、ETSI TS 101 733（技術的に RFC 5126 と同等）で仕様化されており、CMS にいくつかの機能を追加したものです。最も重要な点として、これは、証明書置換と呼ばれる脅威シナリオに対して、署名用証明書への参照を署名内へ含める（***signing-certificate-v2*** 属性を用いて）ことによって防御します。PDF 内の CAdES 署名は、署名辞書内において ***ETSI.CAdES.detached*** サブフィルタを必要とします。

pCOS　CAdES 署名は pCOS で ***signaturefields[...]/cades=true*** として報告されます。

各種 PAdES 署名　PAdES（***PDF Advanced Electronic Signatures*** ＝ PDF 高度電子署名）は ETSI TS 102 778 で仕様化されています。これは、PDF 1.7（ISO 32000-1）で定義されている通りに、PDF 署名に各種オプションと各種制約を追加することによって、CAdES を PDF に適用します。PAdES ではさらなる各種 PDF データ構造の仕様も定めており、これらは PDF 2.0（ISO 32000-2）に盛り込まれています。PAdES はいくつかのパートから成ります（ここで関係のないパート群は省略しています）。

- PAdES パート 2 は、ETSI TS 102 778-2 PAdES Basic 内で仕様化されています。これは、CMS に基づいており、ISO 32000-1 に準拠していますが、その署名の強度を高めるために、そのいくつかのオプショナルな機能を禁じています。たとえば PAdES-Basic は、オリジナルの ISO 32000-1 の定義の中のセキュリティギャップを埋めるために、その**バイトレンジ**が文書全体を対象とすることを必須としています。
- PAdES パート 3 は、ETSI TS 102 778-3 PAdES Enhanced 内で仕様化されています。これは CAdES に基づいており、BES（***Basic Electronic Signature*** ＝基本電子署名）と EPES（***Explicit Policy-based Electronic Signature*** ＝明示的ポリシーベース電子署名）という 2 種のプロファイルを定義しています。EPES は BES を、署名にポリシー識別子と、オプショナルな関与種別表出とを追加することによって拡張しています。この ***policy*** 属性は、その署名がいかなる署名ポリシーのもとで作成されたかを指定します。この ***commitment-type*** 属性を、署名辞書内の ***Reason*** エントリのかわりとして使うこともできます。CAdES では、**「発信証明」・「受取証明」・「認可証明」**等一連の汎用の関与種別を定義しています。
- PAdES パート 4 は、ETSI TS 102 778-4 PAdES Long-Term 内で仕様化されており、長期検証のための手段を提供します。長期検証については 7.6 節「長期検証（LTV）」（127 ページ）で詳しく説明します。パート 4 では、文書レベルタイムスタンプと DSS（7.3.3

1. ETSI の規格群は www.etsi.org/standards から無償で入手可能です。

節「文書セキュリティストア（DSS）」（110 ページ）参照）を導入しています。PAdES パート 4 内で定義されている各種概念は、PAdES パート 2・パート 3 署名に適用できます。すなわち、PAdES-LTV は CMS または CAdES に基づくことができます。

各種 PAdES 署名レベル　署名のライフサイクルを網羅する意図のもと、PAdES 署名には数種類のレベルが定義されています。以下の各種 PAdES 基本署名レベルが ETSI 319 142-1 で仕様化されています：

- 基本署名：PAdES レベル B-B は、PAdES 署名の基礎ブロックです。これは、署名ポリシー識別子を持つ署名と持たない署名に、すなわち EPES と BES に対応しています。このレベルは、短期署名のためのプロファイルと考えることができます。
- 時刻付き署名：PAdES レベル B-T は、PAdES レベル B-B に、ある特定の日時にその署名が存在したことを証明するための署名タイムスタンプを追加しています。
- 長期検証資料付き署名：PAdES レベル B-LT は、PAdES レベル B-T に、検証資料の長期可用性を確保するための検証データを追加しています。証明書チェーンのすべての証明書と、OCSP 応答群と CRL 群を利用可能ですので、この署名は、長い期間の後にも、たとえその CA が利用可能でなくなっていた場合であっても、検証が可能です。
- 検証資料の長期可用性・完全性を実現する署名：PAdES レベル B-LTA は、PAdES レベル B-LT に、検証資料の長期可用性・完全性を確保するための文書レベル（アーカイブ）タイムスタンプと、関連する検証データを追加しています。暗号アルゴリズムか鍵長がもはや充分に強固ではないと考えられるようになった時点等においてタイムスタンプを必要に応じ繰り返し追加することも可能です。LTA 署名は eIDAS 規則の要件に準拠しています。

ETSI EN 319 142-2「パート 2：拡張 PAdES 署名」は、拡張された署名プロファイル群を定義しています：

- PAdES レベル E-BES は、PDF における電子署名のための基本的要件を指定しています。
- PAdES レベル E-EPES は、PAdES E-BES の上に構築されています。これは、署名ポリシー識別子と、オプショナルな関与種別表出を追加しています。
- PAdES レベル E-LTV は、E-BES か E-EPES の上に構築されています。これは、文書セキュリティストア（DSS）と文書タイムスタンプを追加しています。これを利用して、既存の署名を拡張してその署名の長期有効性を保持することが可能です。

Acrobat における CAdES・PAdES 対応　デフォルトでは Acrobat 署名は PAdES パート 2（PAdES-Basic）に準拠しています。Acrobat XI/DC は、CAdES のために以下のように構成

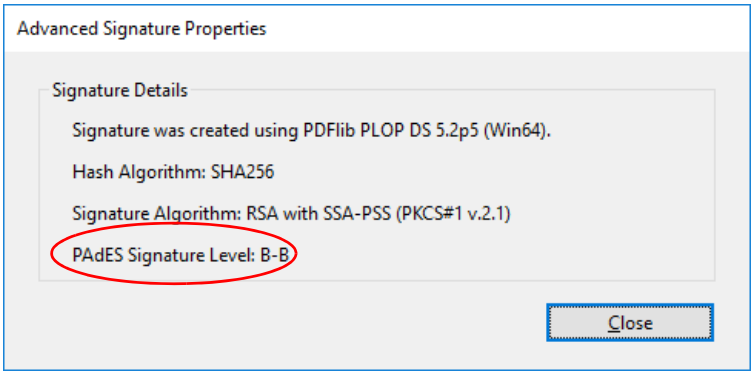


図 7.12
Acrobat DC における PAdES ステータス

すれば、PAdES パート 3 の BES 署名を作成できます：**「編集」→「環境設定」→「署名」→「作成と表示方法」→「詳細 ...」→「デフォルトの署名形式：*CAdES* 相当」**
ポリシー識別子への対応はありませんので、PAdES E-EPES を Acrobat で作成することはできません。しかし、E-BES と E-EPES は両方とも Acrobat で検証できます。
Acrobat XI/DC は、PAdES パート 4 に以下の機能で対応しています：

- 長期検証ステータス情報。詳しくは「Acrobat における LTV ステータス」（127 ページ）を参照してください。
- 署名を LTV 対応に。**「署名」**パネルを開き、**「オプション」**メニューで**「検証情報の追加」**をクリックすることによって可能です。
- 文書レベルタイムスタンプ。

Acrobat DC の 2016 年 10 月のリリース（より正確には Acrobat DC Classic 2015.006.30243 と Acrobat DC Continuous 2015.020.20039）から、署名を右クリックして**「署名のプロパティを表示 ...」→「詳細プロパティ ...」**を選択することによって PAdES 署名レベルを表示させることができるようになりました（図 7.12 参照）。

7.7.2 PLOP DS を用いた PAdES 署名

PLOP DS は、上述の作成者署名と認証署名のためのすべての PAdES 署名レベルに対応しています。署名種別 CMS か CAdES を選択するには ***sigtype*** オプションを用います。各種 PAdES 署名レベルのための機能については追加のオプションで有効にします。デフォルトでは、PLOP DS は、PAdES レベル B-B に準拠した CAdES 署名を生成します。PLOP DS で各種 PAdES 署名レベルを実現するために必要なオプションを表 7.7 に挙げます。

注記 `engine=mscapi` **の場合には PAdES パート 3・パート 4 には対応していません。**

表 7.7 ETSI EN 319 142-1 に従った PAdES 署名レベル

PAdES 署名レベル	PLOP DS のオプション
PAdES レベル B-B。 E-BES（基本電子署名）・ E-EPES（明示的ポリシー電子署名）を含んでいます	PAdES E-BES：`sigtype=cades`（デフォルト） PAdES E-EPES：PAdES E-BES と同様に加え、`policy`
PAdES レベル B-T（署名存在に対する信頼済み時間）	PAdES レベル B-B と同様に加え、`timestamp` を `critical=true` で
PAdES レベル B-LT（長期）	PAdES レベル B-T と同様に加え、`ltv=full rootcertfile/rootcertdir`[1]
PAdES レベル B-LTA（アーカイブタイムスタンプ付き長期）（適格 eIDAS 署名に対しては必須）	PAdES レベル B-LT と同様に加え、`doctimestamp`
PAdES レベル E-:LTV（長期検証）	PAdES レベル B-LTA と同様に加え、`dss=true`、既存の署名を持つ文書に適用

1. LTV ステータスを実現するためにこの他にも、`certfile`・`ocsp`・`crl` といったオプションが必要になる場合があります。7.6.2 節「PLOP DS を用いた LTV 対応署名」（128 ページ）を参照。

PAdES のオプションリストの例　以下の署名オプション（その他 ***digitalid*** といった関連オプション群に加えて）は、PAdES E-BES に従った署名を作成します（これはデフォルト設定ですので省略することも可能です）：

```
sigtype=cades
```

以下の部分的な署名オプションリストは、PAdES E-EPES に従った署名を作成します（架空の署名ポリシー識別子を用いています）：

```
policy={oid=2.16.276.1.89.1.1.1.1.3 commitmenttype=origin}
```

以下の部分的な署名オプションリストは、PAdES レベル B-T に従った署名を作成します：

```
timestamp={critical source={url={http://timestamp.acme.com/tsa}}}
```

以下の部分的な署名オプションリストは、PAdES レベル B-LT に従った署名を作成します：

```
timestamp={critical source={url={http://timestamp.acme.com/tsa}}} ltv=full
```

以下の部分的な署名オプションリストは、PAdES レベル B-LTA に従った、追加のアーカイブタイムスタンプを有するタイムスタンプ付き署名を作成します：

```
ltv=full timestamp={critical source={url={http://timestamp.acme.com/tsa}}} ↵
doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

以下の部分的な署名オプションリストを用いると、既存の PAdES レベル B-LT 署名を、PAdES レベル B-LTA に従った追加のアーカイブタイムスタンプを用いて拡張することができます：

```
ltv=full doctimestamp={source={url={http://timestamp.acme.com/tsa}}}
```

8 PLOP・PLOP DS ライブラリ API リファレンス

8.1 オプションリスト

オプションリストは、PLOP の操作を制御する強力かつ簡単な方式です。多くの API メソッドは、大量の関数引数を必要とするのでなく、オプションリスト（略して optlist）に対応しています。これは、任意の数のオプションを含むことのできる文字列です。オプションリストはさまざまなデータ型や、配列のような複合データに対応しています。多くの言語においてオプションリストは、必要なキーワードと値を連結することによって、簡単に組み立てることができます。C プログラマはオプションリストを組み立てるために、*sprintf( )* 関数を使いたいところでしょう。1 個のオプションリストは、次の形の対を 1 つないし複数含みます。

名前 値（複数可）

名前と値の間、および複数の名前／値ペアどうしの間は、任意の空白類文字（スペース・タブ・キャリッジリターン・ニューライン）で区切ることができます。値は、複数の値のリストから成る場合もあります。また、名前と値の間は等号「=」で結ぶこともできます：

名前=値

単純値　単純値は、以下のデータ型のいずれかを用いることができます：

- 論理値：***true*** または ***false***。論理値のオプションで値が省略されたときは、値 ***true*** と見なされます。略記として、**名前** ***false*** のかわりに ***no*** **名前**を用いることも可能です。
- 文字列：空白類文字または「=」キャラクタを含む文字列は、***{*** と ***}*** でかこむ必要があります。空文字列は {} で作れます。キャラクタ ***{・}・*** は、文字列の中身としたいなら、前に **** キャラクタを付ける必要があります。
- テキスト文字列：いくつかのオプションで用いられる特殊な文字列です。文字列型のオプションの多くは ASCII 値しか受け入れることができませんが、テキスト文字列は ASCII 以外に Unicode 値も保持することが可能です。Unicode 対応の言語バインディング（「各言語バインディングにおける Unicode 対応」（138 ページ）を参照）では、単に任意の Unicode 値をそうしたオプションに与えることができます。Unicode 非対応の言語バインディングでは、文字列を UTF-8 として解釈するべきなら、ユーザーはテキスト文字列の頭に UTF-8 BOM を付ける必要があります。UTF-8 BOM がないときは、テキスト文字列は ***auto*** エンコーディングで、すなわち Windows の場合はカレントコードページ、zSeries の場合は ***ebcdic***、Unix・macOS の場合は ***iso8859-1*** で解釈されます。
- キーワード：固定されたキーワードの定義済みリストのうちの 1 つ
- 浮動小数点値・整数：10 進の浮動小数点値または整数。小数点としては点とカンマが使えます。
- ハンドル：いくつかの内部オブジェクトハンドル、たとえば文書やページのハンドル。実際にはこれらは整数値です。

型によって、またオプションの解釈によっては、さらなる制約が課される場合があります。たとえば、整数や動小数点値は特定の値範囲に制限されるかもしれませんし、ハンドルはそのオブジェクトの種別に対して有効でなければならない等です。オプションに対す

る制約条件は、それぞれの関数の説明に記してあります。単純値のいくつかの例（1 行目は空白キャラクタを含む文字列の例です）：

```
password={secret string}
linearize=true
```

リスト値　リスト値は複数の値から成り、それらの値は単純値かあるいはまたリスト値かもしれません。リストは *{* と *}* でかこまれます。リスト値の例：

```
permissions={ noprint nocopy }
```

注記 バックスラッシュ \ キャラクタは、多くのプログラミング言語において、特殊な取り扱いが必要です。

クォートされていない文字列値　以下の状況においては、オプション値内の実キャラクタが optlist 文法キャラクタと衝突するおそれがあります：

- パスワードまたはファイル名が、開閉照応しない中括弧を、また、バックスラッシュ等特殊キャラクタを含んでいる可能性があります。
- オプションリスト内の和文 SJIS ファイル名（Unicode 非対応言語バインディングにおいてのみ問題となります）

任意のテキストまたはバイナリデータを与えるための、オプションリスト文法要素と干渉しないシンプルな仕組みを実現するために、クォートされていない値は、以下の文法で、長さ指定子とともに与えることも可能です：

```
キー [n]=値
キー [n]={値}
```

ここで 10 進数値 ***n*** は以下を表します：

- Unicode 対応言語バインディングでは：UTF-16 コード単位の数
- Unicode 非対応言語バインディングでは：その文字列を構成するバイト数

この文字列値の左右の中括弧はオプショナルですが、強く推奨します。スペース等区切りキャラクタで始まる文字列には必須です。この文字列内の中括弧・区切りキャラクタ・バックスラッシュは、特殊解釈一切なくリテラルに解釈されます。

スペース・中括弧キャラクタを含むキャラクタ 7 個のパスワードを指定している例。文字列全体を挟んでいる中括弧はこのオプション値の一部ではありません：

```
password[7]={ ab}c d}
```

長方形　長方形は、長方形の左下隅と右上隅の ***x*・*y*** 座標を指定した 4 個の浮動小数点値のリストです。これらの座標は、デフォルト PDF 座標系において、すなわち、ページの左下隅を原点として、ポイントをユニットとして解釈されます。例：

```
rect={ 100 100 200 150}
```

adaptキーワードを用いて、変形なしの自動的な寸法算出を行わせることもできます。7.3.1 節「グラフィックかロゴを用いて署名を視覚化」（105 ページ）を参照してください。

各言語バインディングにおける Unicode 対応　プログラミング言語または環境が Unicode 文字列にネイティブに対応しているとき、そのバインディングを Unicode 対応と呼びます。以下の言語バインディングが Unicode 対応です：

- C++
- .NET
- Java
- Objective-C
- Python
- RPG

これらの環境における文字列の扱いはストレートです：すべての文字列は、ネイティブな UTF-16 形式の Unicode 文字列として与えられます。言語ラッパは、クライアントによって与えられた Unicode 文字列を正しく処理して、自動的にオプションを設定します。

以下の言語バインディングはデフォルトで Unicode 非対応です：

- C（ネイティブな文字列データ型がありません）
- Perl
- PHP
- Ruby

Unicode 非対応言語バインディングに対しては UTF-8 の使用を推奨します。API の仕様の一部は、Unicode 対応言語バインディングと Unicode 非対応言語バインディングとで異なっています。そのような相違については本章の該当箇所の API 解説において示します。

PLOP_convert_to_unicode() 関数を用いると、UTF-8・UTF-16・UTF-32 文字列間の変換と、任意のエンコーディングから Unicode への変換を、行うことができます。その際に BOM を付けることもできます。

8.2 一般関数

C ***PLOP *PLOP_new(void)***

新規の PLOP コンテキストを作成します。

戻り値 新規コンテキストへのハンドル、または十分なメモリが得られない場合は NULL。コンテキストは、他のすべての API メソッドに与える必要があります。

バインディング オブジェクト志向言語では、新規 PLOP オブジェクトが作成されたときには自動的に呼び出されるので、得られません。

Java ***void delete( )***

C# ***void Dispose( )***

C ***void PLOP_delete(PLOP *plop)***

PLOP コンテキストを削除し、その内部リソースをすべて解放します。

詳細 コンテキスト内のすべての開いている文書は自動的に閉じられます。しかし、文書が必要なくなった時点で ***PLOP_close_document( )*** でそれを閉じておくのは良いプログラミング習慣です。

バインディング C の場合、この関数は ***PLOP_TRY( )/PLOP_CATCH( )*** 節の中で呼び出してはいけません。

Java の場合、このメソッドは PLOP のファイナライザメソッドによって呼び出されます。しかし、明示的に ***delete( )*** を呼び出して確かなクリーンアップを行わせることを強く推奨します。例外が起きたときにもこれは然りです。

Perl・PHP の場合、この関数は PLOP オブジェクトが破壊されたときに自動的に呼び出されます。

.NET の場合、非マネージのリソースをクリーンアップするために処理の最後で ***Dispose( )*** を呼び出すべきです。

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)

メモリ上で与えられたデータから、名前付きの仮想の読み取り専用のファイルを作成します。

filename （名前文字列）仮想ファイルの名前。これは任意の文字列であり、以後、他の PLOP 読み出しの中でこの仮想ファイルを参照するために用いることができます。

len （C 言語バインディングのみ）***filename*** の UTF-16 文字列に対する長さ（バイト単位）。***len=0*** の場合、ヌル終端文字列を与える必要があります。

data 仮想ファイルにしたいデータ。C・C++ の場合、これはメモリ位置へのポインタです。Java の場合、これはバイト配列です。Perl・PHP の場合、これは文字列です。

size （C・C++ のみ）データを含むメモリブロックのデータ長をバイト単位で表したもの。

optlist 表 8.1 に従ったオプションリスト。次のオプションが使えます：*copy*。

詳細 この関数は、繰り返し使用される電子 ID や XMP メタデータのために役立つでしょう。仮想ファイルは、入力ファイルを用いるあらゆる API メソッドに与えることができます。こうした関数のなかには、データが必要なくなるまで仮想ファイルにロックをかけるものもあります。仮想ファイルは、***PLOP_delete_pvf( )*** で明示的に、または ***PLOP_delete( )*** で自動的に削除されるまでメモリ上に保持されます。

PLOP オブジェクトはそれぞれ、独自の PVF ファイルの集合を保持します。仮想ファイルは、異なる PLOP オブジェクト間で共有することはできません。別々の PLOP オブジェクトを使用しているマルチスレッドは、PVF の使用を同期する必要はありません。*filename* が既存の仮想ファイルを参照しているときは、例外が発生します。この関数は、*filename* がディスク上の通常のファイルですでに使用されているかどうかはチェックしません。

copy オプションを与えていない限り、対になる ***PLOP_delete_pvf( )*** への呼び出しが成功するまでは、与えたデータを呼び出し側で変更したり解放（削除）したりしてはいけません。このルールに従わないと、クラッシュする可能性が高いです。

表 8.1 PLOP_create_pvf( ) に対するオプション

オプション	説明
copy	（論理値）true の場合、PLOP は、与えられたデータの内部コピーを作ります。この場合、与えたデータをこの呼び出しの直後に呼び出し側で破棄してもかまいません。デフォルト：C・C++ では false、しかしそれ以外のすべての言語バインディングでは true

---

C++ Java C# ***int delete_pvf(String filename)***

Perl PHP ***int delete_pvf(string filename)***

C ***int PLOP_delete_pvf(PLOP *plop, const char *filename, int len)***

名前付きの仮想ファイルを削除し、そのデータ構造を解放します。

filename （名前文字列）***PLOP_create_pvf( )*** に与えたのと同じ、仮想ファイルの名前。

len （C 言語バインディングのみ）*filename* の UTF-16 文字列に対する長さ（バイト単位）。*len=0* の場合、ヌル終端文字列を与える必要があります。

戻り値 対応する仮想ファイルが存在しているがロックされているときは -1（PHP では 0）、それ以外のときは 1。

詳細 ファイルがロックされていなければ、PLOP はただちに、*filename* に関連付けられていたデータ構造を削除します。*filename* が有効な仮想ファイルを参照していないときは、この関数は無言のまま何もしません。この関数への呼び出しが成功した後は、*filename* は再利用することもできます。すべての仮想ファイルは ***PLOP_delete( )*** で自動的に削除されます。

具体的な動作は、対応する ***PLOP_create_pvf( )*** を呼び出したときに *copy* オプションを与えていたかどうかで異なります。すなわち、*copy* オプションを与えていた場合は、ファイルの管理データ構造もファイル内容自体（データ）も両方解放されますが、そうでなかった場合は内容は、クライアント側で解放されるものと思われますので解放されません。

C++ Java C# ***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)***

仮想ファイルか PDFlib 仮想ファイルシステム（PVF）の諸特性を取得します。

filename （名前文字列）仮想ファイルの名前。***keyword=filecount*** のときは、filename は空にすることができます。

len （C 言語バインディングのみ）***filename*** の UTF-16 文字列に対する長さ（バイト単位）。***len=0*** の場合、ヌル終端文字列を与える必要があります。

keyword 表 8.2 に従ったキーワード。

表 8.2 PLOP_info_pvf( ) に対するキーワード

キーワード	説明
exists	ファイルが PDFlib 仮想ファイルシステム内に存在するなら（かつ削除されていないなら）1、しないなら 0
filecount	カレント PLOP オブジェクトのために保持されている PDFlib 仮想ファイルシステムの中のファイルの総数。filename 引数は無視されます。
iscopy	（存在している仮想ファイルに対してのみ）指定した仮想ファイルが作成された際に copy オプションが与えられたなら 1、そうでないなら 0。
lockcount	（存在している仮想ファイルに対してのみ）指定した仮想ファイルに対して PLOP 関数群によって内部的に設定されたロックの数。ロックカウントが 0 のときにのみファイルは削除できます。
size	（存在している仮想ファイルに対してのみ）指定した仮想ファイルのサイズをバイト単位で。

戻り値 ***keyword*** によって要求された通りの何らかのファイルパラメータの値。

詳細 この関数は、仮想ファイルか PDFlib 仮想ファイルシステム（PVF）のさまざまな特性を返します。特性はキーワードで指定されます。

8.3 入力関数

C++ Java C# ***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)***

PDF 文書（保護されているかもしれない）を処理するために開きます。

filename 開きたい PDF ファイルのフルパス名。このファイルは ***SearchPath*** リソースを用いて検索されます。

Unicode 非対応言語バインディングでは、このファイル名は、***filenamehandling*** オプションに従って UTF-8 へ変換されます（***filenamehandling=unicode*** か、与えられたファイル名が UTF-8 BOM で始まっているのでない限り）。***len*** が 0 以外の場合には（C 言語バインディングのみ）、このファイル名は、オプション ***filenamehandling*** にかかわらず、UTF-16 から UTF-8 へ変換されます。このファイル名が変換できないとき、またはこのファイル名が有効な UTF-8 か UTF-16 を構成していない場合には、エラーが発生します。

Windows の場合、必要な権限があれば（ASP で動作している場合はないかもしれない）、UNC パスまたは割り当てられたネットワークドライブも使えます。

len （C 言語バインディングのみ）***filename*** の UTF-16 文字列に対する長さ（バイト単位）。***len=0*** の場合、ヌル終端文字列を与える必要があります。

optlist 表 8.3 に従ったオプションリスト（8.1 節「オプションリスト」（137 ページ）参照）。

戻り値 エラーの場合は −1（PHP では 0）、そうでないなら文書ハンドル。エラーの後は、***PLOP_get_errmsg()*** を呼び出して、そのエラーについてより詳しく知ることを推奨します。

詳細 文書ハンドルは以下の目的に使用できます：

- 入力文書として、さらに ***PLOP_create_document()*** を用いて処理するために使用。
- 署名の体裁としてページを提供（署名オプション ***field*** とサブオプション ***visdoc***）。
- pCOS を用いて文書情報をクエリ。

文書が暗号化されている場合は、そのユーザーパスワードかマスターパスワードを（あるいは、証明書セキュリティの場合には、然るべきデジタル ID を）与える必要があります。ただし、***requiredmode*** オプションが指定されている場合はこの限りではありません。

表 8.3 PLOP_open_document*( ) に対するオプション

オプション	説明
digitalid	（オプションリスト。証明書セキュリティを用いて保護されている入力文書に対してのみ）文書が暗号化されている対象の受信者のデジタル ID。表 8.8 のサブオプション群を用いてデジタル ID を指定します。engine=mscapi の場合には、この digitalid オプションはなしか空でもよく、その場合、デフォルト証明書ストア内にあるすべての ID が試されます。
engine	（キーワード。証明書セキュリティを用いて保護されている入力文書に対してのみ）その文書を復号するための暗号エンジン（デフォルト：builtin）： ***builtin*** 内蔵暗号エンジンを使用します。digitalid で仮想またはディスクファイルを指定する必要があります。 ***mscapi*** （Windows のみ）Microsoft Crypto API を暗号エンジンとして使用します。デジタル ID は、証明書ストアかディスクファイルで与えることができます。digitalid オプションはなしか空かでも可です。
inmemory	（論理値。PLOP_open_document( ) のみ）true の場合、PLOP はファイル全体をメモリ内に読み込んで、そこでそれを処理します。これはシステムによっては（特に MVS）非常なパフォーマンス向上につながりますが、かわりにメモリを食います。false の場合、文書の部分部分が必要に応じて都度都度ディスクから読まれます。デフォルト：false
password	（文字列。暗号化された文書に対しては、requiredmode がある場合以外は必須） パスワードセキュリティを用いて保護されている文書に対して：文書のユーザーパスワードかマスターパスワード。表 5.2（67 ページ）で述べたように、文書にどの操作をしたいかによって、その文書のユーザーパスワードが必要か、マスターパスワードが必要か、それともパスワードが必要ないかが決まります。EBCDIC プラットフォームではパスワードは ebcdic エンコーディングか EBCDIC-UTF-8 で与える必要があります。update=true の場合には、これと同じパスワードが、生成される出力文書のためのマスターパスワードとして使用されます。 digitalid が与えられている場合：証明書セキュリティを用いて保護されている文書に対して必要なデジタル ID に対するパスワード・パスフレーズ・PIN のいずれか。engine=builtin に対しては、password か passwordfile のどちらか一方のみが必ず必要です。それ以外のエンジンではその他の方式が使用できる可能性もあります。EBCDIC プラットフォームでは、このパスワードは ebcdic エンコーディングと前提されます。
passwordfile	（文字列。engine=builtin に対しては、password か passwordfile のどちらか一方のみが必ず必要です。それ以外のエンジンではその他の方式が使用できる可能性もあります）このファイルの先頭行（行末キャラクタないしキャラクタ群を除いて）が、デジタル ID に対するパスワード・パスフレーズ・PIN のいずれかとして使用されます。EBCDIC プラットフォームでは、このパスワードファイルの内容は ebcdic エンコーディングと前提されます。
repair	（キーワード）破損した PDF 入力文書をどう扱うかを指定します。文書を修復すると通常の処理より時間がかかりますが、ある種の破損 PDF の処理ができるようになる可能性があります。ただし文書によっては、修復できないほど破損していることもありえます（デフォルト：auto）： ***force*** 文書に問題があろうとなかろうと、無条件で文書の修復を試みます。 ***auto*** PDF を開く際に問題が検出された場合のみ文書を修復します。 ***none*** 文書を修復する試みは行われません。PDF 内に問題があった場合は、関数呼び出しは失敗します。
requiredmode	（キーワード）文書を開く際に受け入れ可能な最低限の pCOS モード（minimum/restricted/full）。求めた pCOS モードより結果の pCOS モードが低かったときは、呼び出しは失敗します。呼び出しが成功した場合、結果の pCOS モードは少なくともこのオプションで指定したものであることが保証されます。ただし、それより高い可能性もあります。たとえば、暗号化されていない文書に対して requiredmode=minimum を指定した場合、結果は full モードになります。デフォルト：full

表 8.3 PLOP_open_document*( ) に対するオプション

オプション	説明
shrug	（論理値）権限制限に従うとその文書を限定 pCOS モードでしか開くことができない場合に、権限制限を無視します（すなわち PDF 処理が許されます）。パスワードセキュリティの場合にはこれは、その文書がマスターパスワードを用いて暗号化されているにもかかわらず、ユーザーパスワード（もしあれば）のみを与えられていることを意味します。証明書セキュリティの場合にはこれは、然るべき受信者デジタル ID が与えられているけれども、その文書がこの ID に対してマスター権限を設定していないことを意味します。権限設定が無視された場合、pCOS 擬似オブジェクト shrug が true に設定されます。デフォルト：false
xmppolicy	（キーワード）入力文書内の無効な文書レベル XMP の扱いを制御します。無効な XMP は、標準識別子を見つけることができないことを暗黙に前提しますので、たとえば PDF/A 文書がそれとして扱われません。使えるキーワード（デフォルト：rejectinvalid）： ***rejectinvalid*** 無効な XMP の場合には、XML パーサエラーメッセージを含む例外を発生させ、処理を停止させます。 ***ignoreinvalid*** （sacrifice={pdfa pdfua pdfvt pdfx} を含意します）無効な XMP を、XMP が存在しないかのように扱います。出力 XMP は、文書情報項目に基づき生成されます。また、XML 解析エラーメッセージを <pdfx:Exception> 要素内に入れ込みます。 ***remove*** （sacrifice={pdfa pdfua pdfvt pdfx} を含意します）入力 XMP を、有効であってもなくても無条件に無視します。出力 XMP は一から生成されます。これは、望ましくないメタデータを削除するのに役立つでしょう。XMP 内の標準識別子（PDF/A などの）は失われます。

C++ ***int open_document_callback(void *opaque, plop_off_t filesize,***
size_t (*readproc)(void *opaque, void *buffer, size_t size),
int (*seekproc)(void *opaque, plop_off_t offset),
wstring optlist)

C ***int PLOP_open_document_callback(PLOP *plop, void *opaque, plop_off_t filesize,***
size_t (*readproc)(void *opaque, void *buffer, size_t size),
int (*seekproc)(void *opaque, plop_off_t offset),
const char *optlist)

PDF 文書を（保護されているかもしれない）、ユーザーが与えた関数で開きます。

opaque 何らかの不透明なデータ構造へのポインタ。*readproc* へ渡されます。PLOP はこのポインタやその背後のデータを使いません。

filesize 文書の長さをバイト単位で。

readproc メモリ位置 *buffer* にある文書の任意の *size* バイトの切れ端を与えることのできなければならないプロシージャ。このプロシージャは、取得したバイト数を返さなければなりません。

seekproc 文書内の位置 *offset* へシークするためのプロシージャ。このプロシージャはエラーが起きたら -1 を、そうでないなら 0 を返さなければなりません。

optlist 表 8.3 に従ったオプションリスト（8.1 節「オプションリスト」（137 ページ）参照）。

戻り値 エラーの場合は -1（PHP では 0）、そうでないなら文書ハンドル。エラーの後は、*PLOP_get_errmsg( )* を呼び出して、そのエラーについてより詳しく知ることを推奨します。

バインディング　C・C++ 言語バインディングでのみ利用可能です。*plop_off_t* 型は *ploplib.h* 内で条件的に定義されています。これは、2GB を超える大きなファイルに対してはオフセット型として 64 ビット値を保持します。アプリケーションは大容量ファイルサポート（LFS）とともにビルドされる必要があります。

C++ Java C# ***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)***

入力・出力文書を閉じます。

doc　*PLOP_open_document*()* で得られた有効な文書ハンドル。

optlist　オプションリスト（目下使われていません）。

詳細　この関数は、*PLOP_delete()* を呼び出すより前に、*PLOP_open_document*()* を用いて開かれている各文書について呼び出す必要があります。これは、与えられたハンドルに紐付いている文書を閉じて、関連するすべてのリソースを解放します。

8.4 出力関数

C++ Java C# ***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)***

PDF 出力文書を、メモリ内またはディスクファイル上に作成します。

filename （名前文字列）生成したい出力ファイルの名前。***PLOP_open_document()*** に与えた入力ファイル名とは異なっている必要があります。これが空文字列のときは出力はメモリ内に生成され、後で ***PLOP_get_buffer()*** で取り出せます。

Unicode 非対応の言語バインディングの場合、***len=0*** のファイル名はカレントシステムコードページで解釈されますが、ただし UTF-8 BOM が頭についているときは、UTF-8 または EBCDIC UTF-8 として解釈されます。

len （C 言語バインディングのみ）***filename*** の UTF-16 文字列に対する長さ（バイト単位）。***len=0*** の場合、ヌル終端文字列を与える必要があります。

optlist 表 8.4 に従ったオプションリスト（8.1 節「オプションリスト」（137 ページ）参照）。

戻り値 エラーの場合は -1（PHP では 0）、そうでないなら文書ハンドル。エラーの後は、***PLOP_get_errmsg()*** を呼び出して、そのエラーについてより詳しく知ることを推奨します。

電子文書が作成される場合、この関数呼び出しは、以下の場合には失敗します：

- タイムスタンプが取得できず、かつ、***critical*** オプションが設定されている。
- 署名チェーンが再検証され、その間に証明書が期限切れになっているか失効させられている。
- ページに収まらない視覚化文書が与えられている。
- 入力文書が破損しており、かつ、署名が更新モードで作成されている。

PLOP_add_recipient() が 1 回ないし複数回呼び出されたにもかかわらずそれらの呼び出しが 1 つも成功しなかった（すなわちすべての受信者証明書が拒否された）場合には、***PLOP_create_document()*** は失敗します。これは、保護されていない出力がうっかり生成されてしまうことを避けるためです。

詳細 この関数を呼び出す前に、***PLOP_open_document*()*** が呼び出されている必要があります。処理させたい文書を ***input*** オプションで与えます。ユーザーパスワードとマスターパスワードについて強いられる制約条件については 5.2 節「PLOP を用いて PDF 文書をパスワード保護」（67 ページ）を参照してください。

PLOP_create_document() は、その署名チェーンを、OCSP 応答が要求されてから期限切れになった等の理由で、再検証することがあります。

証明書セキュリティモードでは、すなわち、***PLOP_add_recipient()*** が、空でない ***certificate*** オプションリストとともに 1 回でも呼び出されていたなら、この関数は、期限切れになっている受信者証明書が 1 つでもあるかどうかをチェックします。もしあれば、その関数呼び出しは失敗します。

表 8.4 PLOP_create_document( ) に対するオプション

オプション	説明
docinfo	（テキスト文字列の対のリスト）出力文書の文書情報項目群を設定します。文書に文書 XMP メタデータがある場合は、標準文書情報項目は XMP へもミラーされます。オプションリスト内のそれぞれのペアには、項目の名前とその値が入っています。与えた情報項目群は、たとえ PDF 2.0 の場合に文書情報辞書が一切書き出されなくても（オプション emitdocinfo を参照）、常に XMP へ同期されます。以下の定義済みキーとカスタムキーを与えることができます（デフォルト：文書情報項目は入力文書からコピーされます）： ***Subject*** 文書のサブタイトル ***Title*** 文書のタイトル ***Author*** 文書の作成者 ***Keywords*** 文書の内容を表すキーワード ***Trapped*** 文書にトラッピングが適用されているかどうかを示します。許される値は True・False・Unknown です。PDF/X 入力の場合、sacrifice オプションに pdfx か pdfvt があるなら、Unknown のみが許されます。 ***Creator*・*CreationDate*・*Producer*・*ModDate*・*GTS_PDFXVersion*・*GTS_PDFXConformance*・*ISO_PDFEVersion*** 以外の任意の名前 ユーザー定義のフィールド名（スペースキャラクタを含んでいてはいけません）。PLOP は任意の数のフィールドに対応しています。1 個のカスタムフィールド名は 1 度だけ与える必要があります。
emitdocinfo	（論理値。PDF 2.0 出力の場合にのみ意味を持ちます）true の場合、文書情報辞書が書き出されます。デフォルト：false
encryption	（キーワード。パスワードセキュリティと証明書セキュリティに対してのみ意味を持ちます）出力文書を保護するために使用させたい暗号化アルゴリズム。 パスワードセキュリティに対しては、すなわち、masterpassword が与えられている場合には、以下のキーワードが使えます（デフォルト：algo11）： ***algo4*** （PDF 2.0 では非推奨）Acrobat 7/8 に従った AES-128、すなわち pCOS アルゴリズム 4 を使用して暗号化。これは必要に応じて出力 PDF バージョンを PDF 1.6 へ上げます。パスワードは Latin-1 キャラクタ群のみを内容とすることができ、32 キャラクタへ切り詰められます。 ***algo11*** Acrobat X/XI/DC に従った AES-256、すなわち pCOS アルゴリズム 11 を使用して暗号化。これは必要に応じて出力 PDF バージョンを PDF 1.7ext8 へ上げます。パスワードは Unicode キャラクタ群を内容とすることができ、127 UTF-8 バイトへ切り詰められます。 証明書セキュリティに対しては、すなわち、PLOP_add_recipient( ) が、空でない certificate オプションリストとともに 1 度でも成功裏に呼び出されているなら、以下のキーワードが使えます（デフォルト：algo10）： ***algo6*** （PDF 2.0 では非推奨）Acrobat 7 に従った AES-128 すなわち pCOS アルゴリズム 6 を使用した証明書セキュリティを用いて暗号化。これは必要に応じて出力 PDF バージョンを PDF 1.6 へ上げます。 ***algo10*** Acrobat 9 に従った AES-256 すなわち pCOS アルゴリズム 10 を使用した証明書セキュリティを用いて暗号化。これは必要に応じて出力 PDF バージョンを PDF 1.7ext3 へ上げます。
input	（PLOP_open_document*( ) を用いて取得された文書ハンドル。必須）処理させたい入力文書
limitcheck	true の場合、PDF/A-1/2/3・PDF/X-4/5 モードにおいて間接 PDF オブジェクト数の上限（8,388,607）が強制されます。デフォルト：true
linearize	（論理値。署名作成または metadata とともに使うことはできません）true の場合、出力文書は線形化されます。MVS システムの場合、このオプションはメモリ内生成（すなわち空の filename 引数）と併用することはできません。デフォルト：false
master-password[1]	（文字列。update=false を強制）文書をパスワード保護するためのマスターパスワード。これが空の場合、マスターパスワードは適用されません。デフォルト：空

表 8.4 PLOP_create_document( ) に対するオプション

オプション	説明
metadata	（オプションリスト。linearize との併用不可）文書の XMP メタデータを与えます。PDF/A・PDF/X 準拠エントリは、この与える XMP の中では許されません。使えるサブオプション： ***filename*** （名前文字列。必須）妥当な XMP メタデータを UTF-8 形式で含むファイルの名前。 ***validate*** （キーワード）与えた XMP メタデータはキーワードによって検証されます（PLOP は入力文書内の XMP メタデータを検証しないことに留意してください）： ***none*** 検証なし ***xmp2004*** XMP 2004 仕様に従って検証 ***xmp2005*** XMP 2005 仕様に従って検証 ***pdfa1*** xmp2004 と同様ですが、それに加えて定義済みプロパティとスキーマの試験と、PDF/A-1 に従って拡張スキーマの検証も行います。 ***pdfa2*** xmp2005 と同様ですが、それに加えて定義済みプロパティとスキーマの試験と、PDF/A-2・PDF/A-3（両規格はメタデータの必要事項が同じです）に従って拡張スキーマの検証も行います。 デフォルト：入力が PDF/A-1 に準拠していてかつ sacrifice オプションに pdfa が含まれていない場合は pdfa1。入力が PDF/A-2 か PDF/A-3 に準拠していてかつ sacrifice オプションに pdfa が含まれていない場合は pdfa2。それ以外なら none
objectstreams	（キーワード。linearize=true の場合と、PDF/A-1・PDF/X-1a/3 モードにおいては、none が強制されます）出力ファイルサイズを劇的に削減する圧縮オブジェクトストリームを生成（デフォルト：all）： ***all*** 文書情報辞書以外のすべての単純オブジェクトを圧縮オブジェクトストリーム内へ書き込み、圧縮相互参照ストリームを生成。 ***none*** 圧縮オブジェクトストリームも圧縮相互参照ストリームも生成しません。 ***xref*** 圧縮相互参照ストリームを生成しますが、それ以外の圧縮オブジェクトストリームを生成しません。
optimize	（キーワード。update=true の場合には無視されます）適用させたい最適化（デフォルト：none）： ***all*** リソース最適化群を適用。 ***none*** 最適化を一切適用しない。これは若干速度を向上させますが、そのかわりファイルサイズは大きくなります。
permissions	（キーワードリスト。masterpassword か証明書セキュリティ必要です。update=true の場合には不可）文書に対する権限制限のリスト。キーワード noprint・nomodify・nocopy・noannots・noassemble・noforms・noaccessible・nohiresprint・plainmetadata を任意の数含むことができます（表 5.3（68 ページ）参照）。 証明書セキュリティモードにおいては、キーワード plainmetadata のみが許されます。それ以外の権限設定は PLOP_add_recipient( ) の permissions オプションで指定できます。 デフォルト：空
recordsize	（整数。z/OS・USS のみ）出力ファイルのレコードサイズ。デフォルト：0（非ブロック出力）
sacrifice	（キーワードのリスト）このオプションを用いると、入力 PDF の特性と求められた操作とが衝突した場合の動作を制御することができます。デフォルトでは PLOP は、衝突を検出したときには一切出力を生成せず、例外を発生させます。しかし、処理を許すために、文書の何らかの特質を放棄させることが可能です。表 8.5 に挙げるキーワードに対応しています：これらは、入力トリガと操作トリガが両方とも真でない限り無視されます。デフォルト：空のリスト、すなわち、衝突が起きた場合、例外が発生し、出力が一切生成されません。
tempdirname	（文字列）PLOP の内部処理に必要な一時ファイルが作成されるディレクトリの名前。空の場合、PLOP は一時ファイルをカレントディレクトリに生成します。このオプションは、tempfilename オプションが与えられているときは無視されます。デフォルト：空

表 8.4 PLOP_create_document( ) に対するオプション

オプション	説明
tempfilename	（文字列。MVS のみ）PLOP の内部処理に必要な一時ファイルのフルファイル名。空の場合、PLOP は一意な一時ファイル名を生成します。PLOP_close_document( ) の後でこの一時ファイルを削除するのはユーザー側の役割です。このオプションを与えた場合、filename 引数は空にしてはいけません。デフォルト：空
user-password[1]	（文字列。masterpassword が必要）文書をパスワード保護するためのユーザーパスワード。これが空かない場合、ユーザーパスワードは適用されません。デフォルト：空

1. AES-256（アルゴリズム 11）の場合には任意の Unicode キャラクタ群を与えることができますが、AES-128（アルゴリズム 4）の場合には Latin-1 キャラクタ群のみを与えることができます。与えたパスワードは、アルゴリズム 11 の場合には 127 UTF-8 バイトに、アルゴリズム 4 の場合には 32 キャラクタに切り詰められます。EBCDIC プラットフォーム上では、パスワードを ebcdic エンコーディングか EBCDIC-UTF-8 で与える必要があります。

表 8.5 PLOP_create_document( ) の sacrifice オプションに対するキーワード

キーワード	説明
encrypted-attachments	（入力トリガ：文書が暗号化されていないが、暗号化されたファイル添付を 1 個ないし複数含んでいる。操作トリガ：暗号化されたファイル添付に対する適切なパスワードが password オプションで与えられていない）このキーワードを与えると、パスワードの得られない暗号化されたファイル添付は削除されます。 正しいパスワードを与えられていない暗号化ファイル添付を内容として持つ文書は、update=true を用いて署名する際には一切処理できません。
fields	（入力トリガ：文書の中に、NeedAppearances=true の、1 個ないし複数の非署名フォームフィールドがある。操作トリガ：update=false を用いて署名）このキーワードを与えると、署名フィールド群を含めすべてのフォームフィールドが削除されます。
pdfa	（入力トリガ：文書が、PDF/A-1・PDF/A-2・PDF/A-3 のうちのいずれかの準拠レベルに準拠している。操作トリガ：オプション visdoc と非互換な視覚化文書を用いた署名作成、または、オプション userpassword・masterpassword・permissions のうちのいずれか、または、署名が、PDF/A-1 の場合には 64K よりも、PDF/A-2/3 の場合には 32K よりも大きくなった）このキーワードを与えると、PDF/A 入力を処理できますが、PDF/A-1 準拠エントリは削除されます。
pdfua	（入力トリガ：文書が PDF/UA-1 に準拠している。操作トリガ：オプション permissions とキーワード noaccessible を用いた暗号化）このキーワードを与えると、もはや PDF/UA に準拠しなくなった出力を生成でき、その PDF/UA 準拠エントリが削除されます。
pdfvt	（入力トリガ：文書が PDF/VT-1 に準拠している。操作トリガ：pdfx と同じ）このキーワードを与えると、PDF/VT 入力を処理できますが、その PDF/VT・PDF/X 準拠エントリは削除されます。
pdfx	（入力トリガ：文書が PDF/X-1a か PDF/X-3/4/5 に準拠している。操作トリガ：docinfo オプション内の Trapped=Unknown と組み合わせた署名作成、または、field オプションの visdoc サブオプションと組み合わせた署名作成、または、オプション userpassword・masterpassword・permissions のうちのいずれか、または、証明書セキュリティモード）このキーワードを与えると、PDF/X 入力を処理できますが、その PDF/X 準拠エントリは削除されます。文書が PDF/VT-1 にも準拠している場合には、その PDF/VT 準拠エントリも削除されます。
signatures	（入力トリガ：文書が 1 個ないし複数の署名を内容として持っている。操作トリガ：update=true を用いて署名を行う以外のすべての操作。その文書が変更を許可しない証明用署名を内容として持っている場合には、update=true を用いて署名を行うこともトリガになります）このキーワードが与えられており、かつ、入力トリガと操作トリガが両方とも真である場合には、無効な署名群を有する出力を生成してしまうことを避けるために、既存の署名群は消去されます（すなわち、署名値が除去されますが、照応するフォームフィールド群が除去されるわけではありません）。入力トリガと操作トリガが両方とも真であり、かつ、sacrifice={signatures} が与えられていない場合には、PLOP_create_document( ) は失敗します。

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)***

出力文書の内容をメモリから全部または一部取り出します。

size C バインディングでのみ必須。返されるバッファの長さが格納されるメモリ位置へのポインタ。

戻り値 出力データの入ったバッファ。クライアント側では、他のいかなる PLOP ライブラリ関数を呼ぶよりも前に、このバッファ内容を消費する必要があります。

詳細 *PLOP_create_document( )* に空のファイル名を与えることによってメモリ内生成を要求していた場合は（そうでないなら出力はファイルへ書き出されます）、PDF 出力はこの関数によってのみ取り出すことができます。*PLOP_get_buffer( )* は、*PLOP_close_document( )* を呼び出すよりも前に呼び出す必要があります。

8.5 証明書セキュリティ

C++ Java C# ***int add_recipient(String optlist)***

Perl PHP ***int add_recipient(string optlist)***

C ***int PLOP_add_recipient(PLOP *plop, const char *optlist)***

出力文書を保護するための受信者証明書を追加します。

optlist 表 8.6 に従って受信者情報を指定するオプションリスト（8.1 節「オプションリスト」（137 ページ）参照）：
certificate・*conformance*・*engine*・*oaephash*・*rsapadding*・*permissions*

戻り値 エラー時は −1（PHP では 0）、そうでないなら 1。エラーの後には、*PLOP_get_errmsg()* を呼び出すことによってそのエラーに関する詳細を知ることを推奨します。その証明書が見つからないとき、あるいは、PDF 文書を暗号化するために使用できない場合には、この関数は失敗します。

詳細 この関数が、空でない *certificate* オプションとともに 1 回でも呼ばれた場合には、出力文書（群）は、指定されたすべての受信者証明書に対して暗号化されます。証明書群は、ディスクベースまたは仮想ファイルで与えることもできますし、Windows 証明書ストアで与えることもできます。この関数は、その文書に対する受信者のリストを構築するために、任意の回数呼び出すことができます。多くの用途例において、その文書の作成者の証明書をその受信者リストの中へ含めることを推奨します。なぜならそうしないと、その作成者がその保護された文書を開くことができなくなるからです。

与えられる証明書は、「受信者証明書の要件」（83 ページ）で説明した条件に合致している必要があります。

PLOP_create_document() を複数回呼び出すことによって、任意の数の文書を、同一の受信者リストに対して暗号化することが可能です。しかし *PLOP_create_document()* の後に *PLOP_add_recipient()* が再び呼ばれた場合には、新しい受信者のリストを作れるよう、それはまず受信者リストを空リストへリセットします。

この関数は、署名サブオプション値 *update=false* を強制します。なぜなら受信者リストは更新モードでは変更できないからです。

表 8.6 PLOP_add_recipient( ) に対するオプション

オプション	説明
certificate	（オプションリスト。必須）文書を暗号化する対象としたい 1 個の受信者証明書を指定。この証明書は、表 8.8 に従ったサブオプション群を用いて指定されます。セキュリティ上の理由から、そのファイルは searchpath 内で検索されません。 空リストは証明書セキュリティを無効にします。これは、証明書ベースの暗号化とそれ以外の処理との間を切り替えるために役立つでしょう。
conformance	（キーワード）生成される暗号化情報の準拠（デフォルト：acrobat）： ***acrobat*** 文書は Acrobat を用いて開くことが可能（必要な Acrobat のバージョンについては表 6.1（80 ページ）を参照）。その受信者証明書が Acrobat で対応されていないアルゴリズムを使用していた場合にはこの関数呼び出しは失敗します。 ***extended*** その受信者証明書が以下の機能のいずれかを使用していたとしても受け入れる。Acrobat を用いて文書を開くことはできない可能性があります： 鍵長が 8 の倍数でない RSA 証明書。 Brainpool 曲線（RFC 5639）を用いた ECC 証明書。 オプション rsapadding=oaep を用いた暗号化。

表 8.6 PLOP_add_recipient( ) に対するオプション

オプション	説明
engine	（キーワード）その受信者の証明書を特定するために使用するべき暗号エンジン（デフォルト：builtin）： ***builtin*** 内蔵暗号エンジンを使用。証明書は仮想またはディスクファイルで与える必要があります。 ***mscapi*** （Windows のみ）Microsoft Crypto API を暗号エンジンとして使用。証明書は証明書ストアかディスクファイルで与えることができます。
oaephash	（キーワード。rsapadding=oaep の場合にのみ意味を持ちます）OAEP で使用させたいハッシュ関数（デフォルト：sha256）：sha1・sha256・sha384・sha512 のいずれか
rsapadding	（キーワード）RSA 鍵配送のためのパディング機構（デフォルト：pkcs#1）： ***pkcs#1*** PKCS#1 v1.5 に従った RSA パディング。この方式は、Acrobat のすべてのバージョンが対応しています。 ***oaep*** RFC 3447 に従った OAEP（最適非対称暗号化パディング）。この方式はセキュリティ上の諸利点を提供します。Acrobat XI/DC は OAEP に対応していませんので、このキーワードはオプション conformance=extended を必須とします。
permissions	（キーワードリスト）その受信者に対する権限制限のリスト。受信者ごとに異なる権限制限を付与することが可能です。このリストは、任意の数の、表 5.3 に従った権限制限キーワード noprint・nomodify・nocopy・noannots・noassemble・noforms・noaccessible・nohiresprint を内容とします。この他に以下のキーワードも使えます： ***nomaster*** 指定されている権限制限キーワード群に従って、印刷・編集・内容抽出を制限し、かつ、文書のセキュリティ設定の変更を防止します。このキーワードが与えられておらず、かつ、これ以外の上述の権限制限キーワードが 1 つも与えられていない場合、その受信者は文書に対して完全な権限を持ちます。これ以外の上述の権限設定キーワードのいずれかを設定することは nomaster を含意します。 キーワード plainmetadata は、ここでは使えず、PLOP_create_document( ) に与える必要があります。なぜならそれは個別の受信者に適用されるのではないからです。 空リストは、その受信者に権限制限が何も適用されないことを意味します。デフォルト：空リスト

8.6 電子署名

注記 電子署名機能は製品 PLOP DS でのみ利用可能です。

C++ Java C# ***int prepare_signature(String optlist)***

Perl PHP ***int prepare_signature(string optlist)***

C ***int PLOP_prepare_signature(PLOP *plop, const char *optlist)***

署名オプション群を用意します。

optlist 表 8.7 に従って署名オプション群を指定したオプションリスト：

- 署名用証明書（デジタル ID）のためのオプション：***digitalid*・*password*・*passwordfile***
- 署名の文脈に関する情報を与えるためのオプション：
  contactinfo*・*location*・*policy*・*reason
- タイムスタンプのためのオプション：***doctimestamp*・*timestamp***
- 署名視覚化のためのオプション：***field***
- 検証情報を与えるためのオプション：
  certfile*・*crl*・*crldir*・*crlfile*・*ocsp*・*rootcertdir*・*rootcertfile*・*validate
- 証明用署名のためのオプション：***certification*・*preventchanges***
- 署名作成の詳細を制御するためのオプション：
  conformance*・*engine*・*ltv*・*signature*・*sigtype
- 署名の詳細を制御するためのオプション：***dss*・*rsaencoding*・*timestampsize*・*update***

戻り値 エラー時には −1（PHP では 0）、それ以外なら 1。エラーの後には、*PLOP_get_errmsg( )* を呼び出して、そのエラーについてもっと詳しく知ることを推奨します。この関数呼び出しは以下の場合には失敗することがあります：

- 署名者のデジタル ID が見つからないか、または、その秘密鍵へアクセスできない。パスワードや PIN が誤っている等が原因です。
- 検証が失敗。有効な OCSP 応答または CRL を取得できず、その照応する *critical* オプションが設定されている等が原因です。
- *ltv=full* または *validate=full* に対する必要事項を満たせない。

PLOP_prepare_ signature( ) への呼び出しが失敗した後には、それと同じオプション群を用いて再度呼び出しを行うことは避けることを推奨します。なぜならこれは、誤ったパスワード／ PIN を何度も与えること等によって PKCS#11 トークンを無効にしてしまう場合があるからです。

詳細 この関数を用いて用意した署名オプション群を使用して、*PLOP_create_document( )* を用いて任意の数の署名を作成することができます。ここで与えた署名オプション群は、再度 *PLOP_prepare_ signature( )* を（異なる署名オプション群かオプション *nosignature* とともに）呼び出すまで、*PLOP_create_document( )* を用いて作成されるすべての署名に対して使用されます。

この署名用意オプションリストは、署名が作成される前の予測不能な時点群において、再度処理されることがあります。特に、多数の署名が作成された後に CRL または OCSP 応答が期限切れであることが見つかったときは、この署名オプション群は、証明書失効情報をリフレッシュするために再度処理されます。

この関数は、それ以前の呼び出しで PKCS#11 セッションが有効であった場合、それを終了させます。PKCS#11 セッションを明示的に終了させる必要がある場合（たとえば他の

スレッドにそのトークンへのアクセスを与えるために）には、この関数をオプション *signature=false* とともに呼び出すことができます。

表 8.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
certfile	（文字列。engine=mscapi の場合には不可）完全な証明書チェーンを検証して埋め込むために必要となる可能性のある PEM エンコーディングの 1 個ないし複数の中間 CA 証明書を内容とするファイルの名前。
certification	（キーワード）指定した証明レベルを持った証明用（作成者）署名を作成。none 以外の値は、証明用証明書を作成し、1 個の文書の中の最初の署名に対してのみ用いるべきです（デフォルト：none）： ***formfilling*** 証明用署名：フォーム記入することと署名を行うこと（Acrobat のメニュー項目からではなく、署名フィールドをクリックすることによって）は許容されます。ページテンプレートから産み出すことによってページを追加する（手作業でページを追加するのではなく）ことも許容されますが、この技法はめったに使われません。その他の変更は署名を無効化します。 ***formsandannotations*** 署名用署名：フォーム記入すること、署名を行うこと、ページ追加すること、コメントを行うこと（すなわち注釈作成・削除・変更）は許容されます。その他の変更は署名を無効化します。 ***nochanges*** 証明用署名：何か変更が行われるとこの署名は無効になります。 ***none*** 通常の認証署名：文書は証明されません。
conformance	（キーワード）生成される署名の準拠（デフォルト：acrobat）： ***acrobat*** 署名を Acrobat で検証できます（必要な Acrobat バージョンについて表 7.1（104 ページ）を参照）。Acrobat 互換でない証明書かオプションが使用されている場合にはこの関数呼び出し失敗します。 ***extended*** 生成される署名が Acrobat で検証できない署名用証明書とオプション群をも受け入れます。以下の機能は conformance=extended を必要とします： Brainpool 曲線のいずれかを用いた ECC 証明書（RFC 5639） 署名オプション ocsp でサブオプション hash の値が sha1 以外
contactinfo	（テキスト文字列。digitalid に対してのみ意味を持ちます）受信者が署名を検証するためにその署名者に連絡をとることができるようその署名者によって提供される情報（電話番号等）。ただし、これは信頼を確立するためのスケーラブルな解決法としては推奨されません。Acrobat 8/9/X はこの連絡先情報を「署名者」タブの「署名のプロパティ」ダイアログ内に表示します。Acrobat XI/DC はこの連絡先情報を表示しません。

表 8.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
crl	（オプションリストかキーワード。crl=none 以外は digitalid に対してのみ意味を持ちます。engine=mscapi の場合には不可）署名用証明書に対する証明書失効リスト（CRL）を取得して、有効な OCSP 応答が得られない場合にはそれを署名か DSS の中へ埋め込みます。使えるサブオプション（デフォルト：{source={ } critical=false}、すなわち、そのデジタル ID の中に CRLdp 拡張があればそれが使用されます）： ***critical*** （論理値）true にすると、署名用証明書に対して有効な CRL が取得できた場合にのみ署名が生成され、それ以外の場合にはエラーが返されて署名は作成されません。このオプションを false にすると、有効な CRL を取得できなかった場合には CRL 埋め込みは黙って無視されます。デフォルト：true ***filename*** （文字列）署名用証明書に対する CRL を DER エンコーディングで内容とするファイルの名前。この filename オプションを与えると、署名用証明書の中の CRLdp 拡張は無視されます。 ***source*** （ネットワークオプションリスト）署名用証明書に対する CRL 配布点を記述したオプションリスト。プロトコル http・https に対応しています。この source オプションの url サブオプションまたはこの source オプション自体を省略することもでき、その場合にはそのデジタル ID の中の CRL 配布点（CRLdp）拡張が情報源として使用されます。この source ネットワークオプションのサブオプションのうち url・httpauthentication 以外はすべて、署名用証明書以外の証明書群に対する CRL 要求にも適用されます。ですので、発動されるすべての証明書に対する CRL 呼び出しにおいて、同一の資格セット（username/password 等）を使用することが可能です。 デジタル ID の中に CRLdp 拡張が存在していない限り、この filename・source オプションのいずれか 1 つを必ず、かついずれか 1 つのみを、与える必要があります。 このオプションを crl=none とすると、たとえ CRLdp 拡張が存在していても CRL をネットワーク経由で取得しません。これはその署名用証明書だけでなく、関与するすべての証明書に対して効力を持ちます。
crldir	（文字列。engine=mscapi の場合には不可）関与する証明書群を検証するために必要となる可能性のある PEM 形式の CRL 群を内容とするディレクトリの名前。そのファイル名については「証明書・CRL ファイルの命名規則」（179 ページ）を参照。
crlfile	（文字列。engine=mscapi の場合には不可）関与する証明書群を検証するために必要となる可能性のある PEM エンコーディングの 1 個ないし複数の CRL を内容とするファイルの名前。
digitalid	（オプションリスト。認証・証明用署名に対しては必須）表 8.8 に従ったサブオプション群を用いて署名者のデジタル ID を指定。使えるサブオプションは、選択したエンジンによって異なります。
doc-timestamp	（オプションリスト。engine=mscapi の場合には不可）信頼済みタイムスランプ局から文書レベルタイムスタンプを生成（builtin エンジンを使用して）。使えるサブオプション：オプション ***timestamp*** を参照
dss	（論理値。engine=mscapi の場合には不可）true にすると、証明書群と失効情報を文書セキュリティストア（DSS）内へ埋め込みます（7.3.3 節「文書セキュリティストア（DSS）」（110 ページ）参照）。それ以外にすると、このデータを署名の中へ埋め込みます。埋め込みタイムスタンプと文書タイムスタンプに対する検証情報は、このオプションにかかわらず常に DSS 内へ埋め込まれます。デフォルト：sigtype=cades の場合、および、既存の DSS を持った入力文書に対しては true、それ以外なら false

表 8.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
engine	（キーワード）署名するために使用させたい暗号化エンジン（デフォルト：builtin）： ***builtin*** 内蔵暗号化エンジンを使用。デジタル ID を仮想またはディスクファイルで与える必要があります。 ***mscapi*** （Windows のみ）Microsoft Crypto API を暗号化エンジンとして使用。デジタル ID を証明書ストアかディスクファイルで与えることができます。 ***pkcs#11*** PKCS#11 インタフェースを使用して暗号トークンからデジタル ID をロードします。そのトークンのための照応する PKCS#11 DLL ／共有ライブラリの名前を digitalid オプションの filename サブオプションで与える必要があります。
field	（オプションリスト。digitalid に対してのみ意味を持ちます）署名を保持するフォームフィールドの座標と内容を表 8.9 のサブオプション群に従って指定。デフォルト：不可視署名が作成されます
location	（テキスト文字列。digitalid に対してのみ意味を持ちます）署名が作成される物理的位置またはホスト名
ltv	（キーワード。engine=mscapi の場合には不可）署名済み文書を長期検証（LTV）のために用意するかどうかを指定（デフォルト：try）： ***full*** （validate=full を暗黙に前提）署名済み文書を LTV 対応にするために完全な検証情報を埋め込みます。LTV ステータスを実現するには通常、rootcertdir・rootcertfile オプションのいずれかが必須です。さらなる証明書・失効情報を与えるためのオプション certfile・ocsp・crl も必要になる場合があります。証明書のための必要な証明書または失効情報を取得できないときにはこの呼び出しは失敗します。 ***none*** 検証情報を埋め込みません。署名済み文書はより小さくなりますが、LTV 対応ではなくなります。 ***try*** 入手できる限り多くの検証情報を埋め込みます。入手可能な証明書と失効情報によって、署名済み文書は LTV 対応になることもあればならないこともあります。
ocsp	（オプションリストかキーワード。engine=mscapi の場合には不可）表 8.10 に従ったサブオプション群を用いて OCSP 処理を構成。デフォルト：{source={ } critical=false}、すなわち、デジタル ID の中に AIA 拡張があればそれを使用します。 このオプションを ocsp=none とすると、たとえ AIA 拡張が存在していても OCSP 応答を取得しません。これはその署名用証明書だけでなく、関与するすべての証明書に対して効力を持ちます。
password	（文字列。空でも許容されます。engine=builtin の場合には、password か passwordfile のいずれか 1 つ、かついずれか 1 つのみが必須。その他のエンジンでは代替方式を使用することもできます）デジタル ID に対するパスワード・パスフレーズ・PIN のいずれかを指定。engine=pkcs#11 の場合には、このオプションはその暗号トークンに対する PIN を内容とする必要があります。ただし、その PIN をそのトークン自体で対話的に入力する必要がある場合（キーボード付きのスマートカードリーダ等）を除きます。EBCDIC プラットフォームではこのパスワードは ebcdic エンコーディングと見なされます。
passwordfile	（文字列。engine=builtin の場合には、password か passwordfile のいずれか 1 つ、かついずれか 1 つのみが必須。その他のエンジンでは代替方式を使用することもできます）ファイルの 1 行目（1 個ないし複数の改行キャラクタを除いて）が、そのデジタル ID に対するパスワード・パスフレーズ・PIN として使用されます。EBCDIC プラットフォームではこのパスワードファイルの内容は ebcdic エンコーディングと見なされます。

表 8.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
policy	（オプションリスト。sigtype=cades の場合にのみ可。reason を指定した場合には不可。PAdES E-EPES に対しては必須）署名を検証するために使用させたい署名ポリシー。使えるサブオプション： ***commitmenttype*** （キーワード）指定したポリシーのスコープの範囲内の署名に紐付けられる関与の種別。使えるキーワード（デフォルト：none）： ***approval*** 署名者はそのメッセージの内容を認証した。 ***creation*** 署名者はそのメッセージを作成した（ただし必ずしも認証したとは限らず、送信したとも限らない）。 ***delivery*** 信頼済みサービスプロバイダは、メッセージを、そのメッセージの受信者がアクセスできるローカルストアの中で伝達した。 ***none*** 署名には関与種別を含めない。 ***origin*** 署名者は、そのメッセージを作成し、認証し、送信したことを認知している。 ***receipt*** 署名者は、そのメッセージの内容を受信したことを認知している。 ***sender*** 署名者は、そのメッセージを送信した（ただし必ずしもこれを作成したとは限らない）。 ***notice*** （テキスト文字列）署名ポリシーの、人に読めるテキスト説明 ***oid*** （文字列。必須）署名ポリシーのオブジェクト ID ***uri*** （文字列）署名ポリシーの URI
prevent-changes	（論理値。certification が none 以外の場合のみ）true にすると、certification オプションを用いて禁止している各種変更（証明用署名を無効にするである各種変更）が Acrobat で防止されます。すなわち、ユーザーインタフェース内の照応するツールが無効にされます。デフォルト：true
reason	（テキスト文字列。digitalid に対してのみ意味を持ちます。policy とともには不可）文書に署名を行う理由
rootcertdir	（文字列。engine=mscapi の場合には不可）証明書チェーンを検証するために必要になる可能性のある PEM エンコーディングの信頼済みルート CA 証明書群を内容とするディレクトリの名前。ファイルの命名規則について「証明書・CRL ファイルの命名規則」（179 ページ）参照。
rootcertfile	（文字列。engine=mscapi の場合には不可）証明書チェーンを検証するために必要になる可能性のある 1 個ないし複数の PEM エンコーディングの信頼済みルート CA 証明書を内容とするファイルの名前。セキュリティ上の理由により、このファイルは searchpath 内で検索されません。
rsaencoding	（キーワード。engine=mscapi の場合には不可）RSA 署名に対する符号化方式（デフォルト：pkcs#1）： ***pkcs#1*** PKCS#1 v1.5 に従った RSA 符号化。この方式は、すべてのバージョンの Acrobat が対応しています。 ***pss*** RFC 3447/RFC 8017 による PSS に従った RSA 符号化。セキュリティ上の諸利点を提供します。
signature	（論理値）false にすると、署名は作成されません。これは、事前に PLOP_prepare_signature( ) を呼び出して署名オプション群を与えておきながらも署名とそれ以外の処理を切り替えたい場合に役立つでしょう。デフォルト：true
sigtype	（キーワード。digitalid に対してのみ意味を持ちます。engine=mscapi の場合には不可）署名種別（デフォルト：cades）： ***cms*** ISO 32000-1 と PAdES パート 2（ETSI TS 102 778-2）に従った CMS ベースの署名 ***cades*** CAdES（ETSI TS 101 733）と RFC 5126 に従った CAdES ベースの署名。これは PAdES パート 3・パート 4 については必須です。

表 8.7 PLOP_prepare_signature( ) に対するオプション

オプション	説明
timestamp	（オプションリストかキーワード。engine=mscapi の場合には不可）署名は、信頼済み時刻認証局（TSA）によって作成された埋め込みタイムスタンプを含みます。使えるサブオプション（デフォルト：{source={ } critical= false}、すなわち、そのデジタル ID の中に TimeStamp 拡張があればそれを使用）： ***critical*** （論理値。文書レベルタイムスタンプに対しては true を強制されます）true にすると、有効なタイムスタンプが取得できる場合にのみ署名が生成されます。そうでない場合にはエラーが返されます。このオプションを false にすると、有効なタイムスタンプ応答を取得できない場合にはタイムスタンプは黙って無視されます。デフォルト：true ***hash*** （キーワード）タイムスタンプ要求を作成するためのハッシュアルゴリズム。その TSA がそのアルゴリズムに対応している必要があります（デフォルト：sha256）：sha1（非推奨）・sha256・sha384・sha512 のいずれか ***policy*** （文字列）TSA ポリシーの OID。そのポリシーのもとでタイムスタンプを作成します。指定したポリシーにその TSA が対応していない場合にはタイムスタンプは失敗します。 ***source*** （表 8.11 に従ったネットワークオプションリスト）TSA を記述したオプションリスト。プロトコル http・https に対応しています。 文書レベルタイムスタンプではなく埋め込みタイムスタンプの場合のみ：この source オプションの url サブオプションまたはこの source オプション自体を省略することもでき、その場合にはそのデジタル ID の中の TimeStamp 拡張が使われます。 キーワード none にすると、たとえ署名用証明書の中に TimeStamp 拡張が存在していてもタイムスタンプを埋め込みません。
timestamp-size	（整数）タイムスタンプオブジェクトの推定サイズ。文書タイムスタンプと署名タイムスタンプの CMS コンテナのための容量を予約するために使用されます。デフォルト：7168
update	（論理値）true にすると、署名データが、1 個ないし複数の増分 PDF 更新セクションとして、元の文書の複製に追加されます。それ以外にすると、PDF オブジェクトヒエラルキーは書き直され、したがって既存の署名群は失われます。埋め込みタイムスタンプと文書タイムスタンプに対する検証情報は、このオプションにかかわらず、常に更新として追加されます。更新モードは、修復を要する入力文書に対しては可能ではありません。 デフォルト：true、ただし暗号化はこのオプションの値を false に強制します（すなわち、PLOP_create_document( ) の masterpassword または userpassword オプション、あるいは、空でない certificate オプションリストとともに PLOP_add_recipient( ) を呼び出し）
validate	（キーワード）関与する証明書群の検証を制御（デフォルト：ltv=full の場合には full、それ以外の場合には formal）： ***formal*** 以下のチェックが行われます： クリティカルな拡張フラグ群・鍵使用方法等がチェックされます。 OCSP 応答が要求された場合には取得され、ステータス「有効」を持った有効な応答を必須とします。 CRL が要求された場合には取得され、署名用証明書が CRL に照らしてチェックされます。CRL の日付がチェックされます。 ***full*** validate=formal の内容に加えて、証明書チェーンの完全な検証。これは、必要なすべてのルートおよび中間 CA 証明書が入手可能であり、かつ、関与するすべての証明書（ルート証明書と id-pkix-ocsp-nocheck 拡張を持つ OCSP レスポンダを除く）に対する OCSP または CRL 失効情報も入手可能であることを必須とします。

表 8.8 PLOP_prepare_signature( )・PLOP_open_document( ) の digitalid オプションの、および、PLOP_add_recipient( ) の certificate オプションのサブオプション

オプション	説明
engine=builtin の場合のサブオプション（digitalid または certificate オプションから呼ぶ場合）：	

表 8.8 PLOP_prepare_signature( ) ・ PLOP_open_document( ) の digitalid オプションの、および、PLOP_add_recipient( ) の certificate オプションのサブオプション

オプション	説明
filename[1]	（文字列。必須）digitalid オプションから呼ぶ場合：PKCS#12 または PFX 形式の、ディスクベースまたは仮想デジタル ID ファイルの名前（変換のヒントは付章 A「証明書を用いる作業」を参照）。 certificate オプションから呼ぶ場合：PEM または DER エンコーディングの、ディスクベースまたは仮想 X.509 証明書ファイルの名前。この証明書ファイルはちょうど 1 個の証明書を内容としている必要があります。
engine=pkcs#11 の場合のサブオプション（digitalid オプションから呼ぶ場合のみ）：	
signercert	（文字列。id か label のどちらか一方のみを必須）PKCS#11 トークン上の照応する秘密鍵に対する公開鍵を内容とする、DER エンコーディングで表された、署名者の証明書の名称。この秘密鍵については、オプション id または label のいずれかを通じて選択する必要があり、さらに、オプション subject を通じて修飾することも可能です。このオプションは、秘密鍵のみを保管していてそれに照応する証明書を保管していない HSM（AWS CloudHSM 等）に対しては必須です。デフォルト：証明書ファイル名なし、すなわち、その証明書が秘密鍵とともにトークン上で得られる必要があります。
externalhash	（論理値）true の場合、署名に対する文書ハッシュは PKCS#11 インタフェースを通じて（すなわちトークン上や HSM 上で）作成され、そうでない場合、内蔵エンジンを用いて作成されます。デフォルト：false
filename	（文字列。必須）暗号トークンに対する PKCS#11 DLL ／共有ライブラリの名前。これは PVF ファイルでなくディスクベースのファイルである必要があります。例：cryptoki.dll
id	（文字列。signercert が必須）署名に用いたい秘密鍵を、その鍵識別子（PKCS#11 の属性 CKA_ID）によって選択。これは 10 進文字列か、16 進文字列（接頭辞 0x とともに）として与える必要があります。例：id=0x03A247B2
issuer	（文字列）デジタル ID を、その issuer フィールドによって（与えたクエリに PKCS#11 属性 CKA_ISSUER が一致）選択。このクエリの形式の説明は後述の subject を参照してください。
label	（文字列）デジタル ID（あるいは signercert を指定している場合には秘密鍵）を、そのユーザーフレンドリーラベル（PKCS#11 属性 CKA_ LABEL）によって選択。
serial	（文字列）デジタル ID を、そのシリアル番号（PKCS#11 属性 CKA_SERIAL_NUMBER）によって選択。このシリアル番号は、10 進文字列か 16 進文字列（頭に 0X を付ける）として与える必要があります。例： serial=0x03A247B2
slotid	（正の整数）トークンをインタフェースするスロットの番号。これを用いると、複数のスロットが利用可能な場合にスロットを直接選択できます。
subject	（文字列）デジタル ID（あるいは signercert を指定している場合には秘密鍵）を、その subject フィールド（PKCS#11 属性 CKA_ SUBJECT）によって選択。このクエリは、形式 /type0=value0/type1=value1/... である必要があります。キャラクタを \（バックスラッシュ）でエスケープすることもできます。属性の順序は意味を持ちます。トークンが複数のデジタル ID を内容としている場合には、オプション issuer ・ label ・ subject を用いて証明書選択が可能です。 例：subject={/C=DE/L=Munich/O=PDFlib GmbH/CN=PLOP Demo Signer RSA-2048}

表 8.8 PLOP_prepare_signature( ) ・ PLOP_open_document( ) の digitalid オプションの、および、PLOP_add_recipient( ) の certificate オプションのサブオプション

オプション	説明
sticky	（論理値）true にすると、PKCS#11 DLL ／共有ライブラリが、処理の終了まで、ロードされたままになります。これは、パフォーマンス上の優位を提供する可能性があり、また、DLL ／共有ライブラリ内の、その初期化ルーチン内のメモリリークといった問題を、回避するために役立つ可能性があります。ただし、他の PKCS#11 DLL ／共有ライブラリを同一プロセス内でロードすることは一切できません。このオプションをひとたび true に設定したら、その後の同一プロセス内の各呼び出しも sticky=true を与える必要があり、また、サブオプション filename は黙って無視されます。false にすると、PKCS#11 ライブラリは、このライブラリを使用している最後の PLOP オブジェクトに対する PLOP_delete( ) への呼び出しでアンロードされます。デフォルト ：false
threadsafe	（論理値）true にすると、PKCS#11 ライブラリはスレッドセーフ操作に対応している必要があり、スレッドセーフモードで初期化されます。PKCS#11 がスレッドセーフ操作に対応していない場合にはこの呼び出しは失敗します。false にすると、PKCS#11 ライブラリはシングルスレッドモードで初期化されます。これはシングルスレッドのアプリケーションに対してのみ許容されます。デフォルト ：true
engine=mscapi の場合のサブオプション（digitalid または certificate オプションから呼ぶ場合）：	
filename[1]	（文字列。PLOP_prepare_signature( ) から呼ぶ場合には filename ・ store のいずれかが必須[2]）PKCS#12 または PFX 形式の、ディスクベースまたは仮想デジタル ID ファイルの名前（変換のヒントについて付章 A「証明書を用いる作業」を参照）。
storelocation	（キーワード）証明書ストアの場所（デフォルト ：current_user）：current_service ・ current_user ・ current_user_group_policy ・ local_machine ・ local_machine_enterprise ・ local_machine_group_policy ・ services ・ users 以下の場所については遠隔で開くことができ、そのためには store オプションの頭にそのコンピュータ名を付けます（1 個のバックスラッシュキャラクタで区切って）：local_machine ・ local_machine_group_policy ・ services ・ users。
subject	（文字列。PLOP_prepare_signature( ) か PLOP_add_recipient( ) から呼んでいる場合、かつ store を指定している場合には必須。そうでない場合には無視されます）subject フィールドの内容が与えられた文字列であるデジタル ID を選択。それは通常、そのデジタル ID の common name（CN）フィールドを保持しています。PLOP_open_document( ) から呼ぶ場合には、PLOP が自動的に適切な ID を見つけますので、このサブオプションは必要ありません。
store	（文字列。PLOP_prepare_signature( ) から呼んでいる場合には filename ・ store のいずれかが必須）証明書ストアの名前。例 ：My ・ Root ・ Trust。 storelocation=services か storelocation=users の場合には、このストア名の頭にサービスまたはユーザー名を付ける必要があります（1 個のバックスラッシュキャラクタで区切って）。デフォルト ：My

1. certificate オプションから呼ばれた場合には、セキュリティ上の理由から、このファイルは、searchpath 内で検索されません。
2. PLOP_prepare_signature( ) で filename も store も与えられていない場合は、デフォルトストア内にあるすべての ID が試されます。

表 8.9 PLOP_prepare_signature( ) の field オプションのサブオプション

オプション	説明
fillexisting	（論理値。文書の中に 1 個ないし複数の署名フィールドがあり、かつ name オプションを与えていない場合にのみ意味を持ちます）true にすると、入力文書内の最初の署名フィールドを用いて署名が行われます。false にすると、パターン Signature# に基づいた一意な名前を用いて新規の署名フィールドが作成されます。PDF/UA モードで visdoc オプションが与えられている場合にはこのオプションは強制的に true になります。デフォルト ：false

表 8.9 PLOP_prepare_signature( ) の field オプションのサブオプション

オプション	説明
name	（テキスト文字列。末尾をピリオド「.」キャラクタにしてはいけません）既存または新規の署名フィールドの名前。文書がこの名前を持った署名フィールドを内容として持っている場合には、それが署名のために使われ（そして page は無視され）、そうでない場合にはそのフィールドが作成されます。この名前を持ったフィールドがあるが種別が署名でない場合にはエラーが発生します。 デフォルト：署名フィールドがない場合には、名前 Signature1 を持った新規署名フィールドが作成されます。そうでない場合には、フィールド作成はオプション fillexisting によって制御されます。
page	（正の整数。既存の署名フィールドへの記入が行われる場合には無視されます）署名フィールドが作成されるページの番号。先頭ページの番号を 1 とします。デフォルト：1
position	（キーワード 2 個のリスト）フィールド内の視覚化ページの相対位置。この視覚化ページは、与えたキーワード群に従ってその長方形内に配置され、その縦横比を保ちながらその長方形内に完全にはめ込まれるように拡縮されます。1 個目のキーワードは横位置を、値 left・center・right のいずれか 1 つを用いて指定します。2 個目のキーワードは縦位置を、値 top・center・bottom のいずれか 1 つを用いて指定します。両方の値が同じの場合には、1 個のキーワードを指定すれば足ります。デフォルト：{center}
rect	（長方形）署名フィールドの左下隅と右上隅の座標を PDF 座標（1 単位は 1/72 インチで左下隅を原点とする）で表したもの。この指定した長方形が視覚化ページで完全に満たされます。変倍を防ぐために 1 個または 2 個の座標のかわりにキーワード adapt を与えることもできます。この場合には、足りない座標（群）が自動的に算出されます。少なくとも 1 つの隅を明示的に指定する必要があります。この長方形はそのページをはみ出してはいけません。はめ込み処理について詳しくは「署名フィールドの位置と寸法」（106 ページ）を参照してください。4 個のゼロ値を用いた空の長方形は不可視フィールドになります。 デフォルト：既存のフィールドが使われる場合にはその長方形がデフォルトとなります。そうでない場合には空の長方形（すなわち不可視署名）。
tooltip	（空でないテキスト文字列）署名フィールドのためのツールチップ（代替テキストともいいます）のテキスト。これはスクリーンリーダによってアクセシビリティを向上させるために利用される可能性があります。デフォルト：なし
visdoc	（PLOP_open_document( ) を用いて取得した文書ハンドル。PDF/X・PDF/VT モードでは不可。空でないフィールド長方形に対してのみ可であり、かつこの場合には必須）ページ上の署名を視覚化するために使用するページを含む文書。PDF/A モードでは、この視覚化文書は、生成される出力に互換である必要があります（「PDF/A 準拠」（107 ページ）参照）。PDF/UA モードでは、入力文書が然るべき署名フォームフィールドを内容として持っている必要があります（「PDF/UA 準拠」（108 ページ）参照）。
vispage	（整数。visdoc を与えた場合にのみ意味を持ちます）署名を視覚化するために使用するページの、その文書内の番号（先頭ページの番号は 1）。デフォルト：1

表 8.10 PLOP_prepare_signature( ) の ocsp オプションのサブオプション

オプション	説明
critical	（論理値。digitalid に対してのみ意味を持ちます）true にすると、署名用証明書に対する有効な OCSP 応答がステータス「有効」を持って返された場合にのみ署名が生成されます。それ以外の場合にはエラーが返され、署名は作成されません。このオプションを false にすると、有効な OCSP 応答を取得できなかった場合には、OCSP 応答の埋め込みは黙って無視されます。デフォルト：true
freshness	（整数）OCSP 応答の thisUpdate エントリの後何分までなら応答を有効と見なすべきか。この指定した時間（maxclockskew によって延長）よりも応答が古い場合には、それは無効と見なされ、使用されません。デフォルト：1440（1 日）

表 8.10 PLOP_prepare_signature( ) の ocsp オプションのサブオプション

オプション	説明
hash	（キーワード）すべての OCSP 要求・応答内で証明書を識別するために使用されるハッシュアルゴリズム。そのアルゴリズムにその OCSP レスポンダが対応している必要があります（デフォルト：sha1）：sha1・sha256・sha384・sha512 のいずれか。 なお、Acrobat XI/DC は sha1 にしか対応していませんので、それ以外のすべての値は conformance=extended を必要とします。
maxclockskew	（整数）OCSP 応答内の thisUpdate エントリと freshness オプションに従ってその応答が充分に新鮮かどうかをチェックする際に、ずれを何分まで許すか。このオプションを用いて、ネットワーク遅延や不正確なシステム時計を補償できます。デフォルト：5
nonce	（論理値）true にすると、ノンス（nonce =「number used only once =数は 1 回だけ使用される」）拡張がすべての OCSP 要求の中へ含められ、かつそれと同じ値が OCSP 応答内に存在している必要があります。ノンス処理は、反射攻撃を防ぎますが、キャッシングを妨げますので、これに対応していない OCSP レスポンダもあります。デフォルト：true
source	（ネットワークオプションリスト）署名用証明書に対する OCSP 応答が要求されるサーバを記述したオプションリスト。この応答はその後、その署名か DSS の中へ埋め込まれます。プロトコル http・https に対応しています。この source オプションの url サブオプションまたはこの source オプション自体を省略することもでき、その場合にはその URL はそのデジタル ID の中の authorityInfoAccess 拡張（AIA）から採られます。 この source ネットワークオプションのサブオプションのうち url・httpauthentication 以外はすべて、署名用証明書以外の証明書群に対する OCSP 要求にも適用されます。ですので、発動されるすべての証明書に対する OCSP 呼び出しにおいて、同一の資格セット（username/password 等）を使用することが可能です。

ネットワークオプションリスト　いくつかの電子署名機能は、TSA や OCSP レスポンダ等ネットワークリソースへのアクセスを必要とします。サーバと、そこへアクセスするための詳細については、表 8.11 に従ったネットワークオプションリストの中で指定されます。表 8.7・表 8.10 のオプションのうち、データ型「ネットワークオプションリスト」を使用するものは、対応しているプロトコルのリストを指定します。ネットワークオプションリストの使用例をいくつか挙げます（ネットワークオプションリストの部分を青で示しています）：

```
timestamp={source={url={http://timestamp.acme.com/}} hash=sha384} digitalid=...

ocsp={source={url={http://ocsp.acme.com/}} } digitalid=...

ocsp={source={url={http://ocsp.acme.com/}
        proxy={url={http://user:password@proxy.company.com:8080}}} } digitalid=...

ocsp={source={timeout=1000}} digitalid=...
```

表 8.11 ネットワークオプションリストに対するサブオプション

オプション	説明
httpauthentication	（キーワード。http に対してのみ）試行させたい認証方式。ある特定の認証方式に（ないしはどの方式にも）サーバが対応していない場合もあります。パフォーマンス上の利点の観点から、認証種別をデフォルトでなく明示的に設定するほうが望ましい場合があります。使えるキーワード（デフォルト：any）： ***any*** サーバが対応している最もセキュアな認証方式を選択。 ***anysafe*** any と同じ内容に加えて、ベーシック認証を除外。 ***basic*** ユーザー名とパスワードを用いたベーシック認証。この方式は、ユーザー名とパスワードがプレーンテキストのままネットワークへ送信されるので推奨されません。 ***digest*** RFC 2617 に基づいてハッシュ化されたユーザー名とパスワードを用いたダイジェスト認証。 ***ntlm*** Microsoft 製品群で使用されているような NTLM 認証
password	（文字列）ベーシック・ダイジェスト認証のためのパスワード
proxy	（オプションリスト）プロキシサーバを介したネットワークアクセスを構成するためのサブオプション： ***httpauthentication*** （キーワード。http プロキシに対してのみ）同名のメインのネットワークオプションを参照。 ***noproxy*** （文字列）プロキシを必要としないホスト名のカンマ区切りリスト。数値 IPv6 アドレスについては、カッコに入れずに指定する必要があります。 ***password*** （文字列）同名のメインのネットワークオプションを参照。 ***sslcertdir*** （文字列。https プロキシに対してのみ）同名のメインのネットワークオプションを参照。 ***sslcertfile*** （文字列。https プロキシに対してのみ）同名のメインのネットワークオプションを参照。 ***sslverifyhost*** （文字列。https プロキシに対してのみ）同名のメインのネットワークオプションを参照。 ***sslverifypeer*** （文字列。https プロキシに対してのみ）同名のメインのネットワークオプションを参照。 ***url*** （文字列。必須）プロキシサーバのホスト名か数値 IP アドレス。この URL にはユーザー名とパスワードを含めることもできます。数値 IPv6 アドレスについては、カッコ [...] に入れる必要があります。コロン「:」を付けてポート番号を末尾に付けることもできます。ポート番号を指定しない場合は、デフォルトポート番号 1080 が使用されます。プロトコルを指定しない場合は http が使用されます。 ***username*** （文字列）同名のメインのネットワークオプションを参照。 プロキシサーバを、共通環境変数 http_proxy・https_proxy・no_proxy・all_proxy を用いて構成することも可能です。オプションは環境変数よりも優先されます。
sslcertdir	（文字列。https に対してのみ）SSL 接続を確立するために必要となる可能性のある PEM エンコーディングの信頼済み CA 証明書群を内容とするディレクトリの名前。ファイル命名規則について「証明書・CRL ファイルの命名規則」（179 ページ）を参照してください。
sslcertfile	（文字列。https に対してのみ）SSL 接続を確立するために必要となる可能性のある PEM エンコーディングの 1 個ないし複数の信頼済み CA 証明書を内容とするファイルの名前。
sslverifyhost	（論理値。https に対してのみ）true にすると、SSL 接続を確立するためにサーバ証明書内の Subject Alternate Name フィールドが URL 内のホスト名と一致することが必須になります。デフォルト：true
sslverifypeer	（論理値。https に対してのみ）true にすると、sslcertdir または sslcertfile オプションを用いて与えた信頼済み証明書の集合に対してサーバ証明書が検証可能であることが必須となります。false にすると、サーバ証明書のための既知の信頼済みルートが一切得られないために検証ができないサーバ証明書であっても受け入れられます。デフォルト：true

表 8.11 ネットワークオプションリストに対するサブオプション

オプション	説明
timeout	（整数）リソースへアクセスする際のタイムアウトをミリ秒単位で指定。値 0 とするとタイムアウトなしになります。デフォルト：15000
url	（文字列。通常は必須ですが、URL が文脈からわかる場合にはオプショナル）頭のプロトコル識別子を含めて完全に修飾されたネットワークリソースの URL。使えるプロトコルの集合は、ネットワークオプションリストを使用する各オプションの説明で指定されています。キャラクタを %20 等 URL エンコーディングで指定することもできます。この URL はユーザー名とパスワードを含むこともできます。例： `http://user:password@timestamp.acme.com/`
username	（文字列）ベーシック・ダイジェスト認証のためのユーザー名

8.7 例外処理

PLOP では、ライブラリの例外を C 言語で取り扱うための追加のメソッドを提供しています。それ以外の PLOP の言語バインディングでは、それぞれの言語のネイティブの例外処理システムを利用しています（*try/catch* 節等）。言語ラッパは、生成される例外オブジェクトの中に、例外の番号・説明・API メソッド名に関する情報を入れ込みます。

PLOP 例外が発生した時には、その PLOP オブジェクトについては ***PLOP_delete()*・*PLOP_get_errnum()*・*PLOP_get_errmsg()*・*PLOP_get_apiname()*** 以外の PLOP 関数は一切呼び出してはいけません。

Java と .NET 用の PLOP 言語バインディングでは別途、***PLOPException*** オブジェクトを定義しており、これは詳細なエラー情報にアクセスするためのメンバをいくつか提供しています。

C++ Java C# ***int get_errnum()***

Perl PHP ***int get_errnum()***

C ***int PLOP_get_errnum(PLOP *plop)***

もっとも最近に発生した例外、ないし失敗した関数呼び出しの原因の番号を得ます。

戻り値 例外のエラー番号。

バインディング .NET の場合、このメソッドは ***PLOPException*** オブジェクトの中の ***Errnum*** としても利用可能です。
Java の場合、このメソッドは ***PLOPException*** オブジェクトの中の ***get_errnum()*** としても利用可能です。

C++ Java C# ***String get_errmsg()***

Perl PHP ***string get_errmsg()***

C ***const char *PLOP_get_errmsg(PLOP *plop)***

もっとも最近に発生した例外、ないし失敗した関数呼び出しの原因の説明テキストを得ます。

戻り値 エラーを説明する文字列、またはもっとも最近の API 呼び出しが何らエラーを発生させなかった場合は空文字列。

バインディング .NET の場合、このメソッドは ***PLOPException*** オブジェクトの中の ***Errmsg*** としても利用可能です。
Java の場合、このメソッドは ***PLOPException*** オブジェクトの中の ***getMessage()*** としても利用可能です。

C++ Java C# ***String get_apiname()***

Perl PHP ***string get_apiname()***

C ***const char *PLOP_get_apiname(PLOP *plop)***

もっとも最近の例外を発生させた、ないし失敗した API メソッドの名前を得ます。

戻り値 PLOP API メソッドの名前。

バインディング　.NET の場合、このメソッドは *PLOPException* オブジェクトの中の *Apiname* としても利用可能です。
Java の場合、このメソッドは *PLOPException* オブジェクトの中の *get_apiname( )* としても利用可能です。

C *PLOP_TRY(PLOP *plop)*

例外処理フレームをセットアップします。必ず *PLOP_CATCH( )* と対にする必要があります。

詳細　「エラー処理」（45 ページ）参照。

C *PLOP_CATCH(PLOP *plop)*

例外をキャッチします。必ず *PLOP_TRY( )* と対にする必要があります。

詳細　「エラー処理」（45 ページ）参照。

C *PLOP_EXIT_TRY(PLOP *plop)*

PLOP_TRY( ) の中から、対応する *PLOP_CATCH( )* 節へ入ることなく抜けることを、例外機構に通知します。

詳細　「エラー処理」（45 ページ）参照。

C *PLOP_RETHROW(PLOP *plop)*

例外を他のハンドラへ投げなおします。

詳細　「エラー処理」（45 ページ）参照。

8.8 グローバルオプション

C++ Java C# ***void set_option(String optlist)***

Perl PHP ***set_option(string optlist)***

C ***void PLOP_set_option(PLOP *plop, const char *optlist)***

PLOP のための 1 つないし複数のグローバルオプションを設定します。

optlist 表 8.12 に従ってグローバルオプションを指定するオプションリスト。1 つのオプションが複数回与えられた場合、最後に出てきたものがそれより前のすべてを上書きします。1 つのオプション（***searchpath*** 等）に対して複数の値を与えたいときは、すべての値を 1 つのリスト引数にしてこのオプションに与えます。

詳細 表 8.12 で特記してあるオプションについては、この関数を複数回呼び出すことで値を蓄積させることができます。特記していないオプションについては、新しい値が古い値を上書きします。

表 8.12 PLOP_set_option( ) に対するグローバルオプション

オプション	説明
filename-handling	（キーワード）ファイル名のエンコーディングを示します。Unicode 非対応言語バインディングにおいて UTF-8 BOM なしで関数引数として与えられたファイル名は、そのファイルシステムにおいて違法となるであろうキャラクタから保護するため、かつそのファイル名の Unicode 版を作成するために、このオプションに従って解釈されます。ファイル名が、ここで指定したエンコーディングの範囲外のキャラクタを含んでいる場合には、エラーが発生します。デフォルト：Windows・macOS では unicode、それ以外では honorlang： ***ascii*** 7 ビット ASCII ***basicebcdic*** コードページ 1047 に従った基本 EBCDIC、ただし Unicode 値 <= U+007E のみ ***basicebcdic_37*** コードページ 0037 に従った基本 EBCDIC、ただし Unicode 値 <= U+007E のみ ***honorlang*** 環境変数 LC_ALL・LC_CTYPE・LANG が解釈されます。LANG で指定されているコード集合がある場合にはそれがファイル名に適用されます。 ***unicode*** （EBCDIC-）UTF-8 形式の Unicode エンコーディング すべての ***8*** ビット・***CJK*** エンコーディングの名前 PLOP によって認識される任意のエンコーディング
license	（文字列）ライセンスキーを設定します。PLOP_open_document*( ) への初めての呼び出しよりも前に設定する必要があります。
licensefile	（文字列）ライセンスキー（複数可）の入ったファイルの名前を設定します。ライセンスファイルは、PLOP_open_document*( ) への初めての呼び出しよりも前に 1 度だけ設定できます。あるいはライセンスファイルの名前は、PLOPLICENSEFILE という環境変数で与えたり、（Windows の場合）レジストリで与えたりすることも可能です。
frontpage	（論理値）false の場合、有効なライセンスキーが見つからないときに例外を発生させます。true の場合、0.1 節「ソフトウェアをインストール」（7 ページ）に従って評価モードで表紙が生成されます。このオプションは、PLOP_open_document*( ) への初めての呼び出しよりも前に設定する必要があります。有効なライセンスキーが見つかったときは、このオプションは何の効果も持ちません。デフォルト：true
logging[1]	（オプションリスト）表 8.14 に従ってログ記録出力を指定したオプションリスト。あるいはログ記録オプション群を。PLOPLOGGING という環境変数で、または Windows ではレジストリから与えることも可能です。空のオプションリストにすると、以前の呼び出しで設定したオプション群を用いたログ記録を有効になります。環境変数が設定されている場合、ログ記録は、PLOP_new( ) への最初の呼び出しの直後に開始されます。

表 8.12 PLOP_set_option( ) に対するグローバルオプション

オプション	説明
mmiolimit	（整数）メモリマップされる入力ファイルのサイズの上限を MB（＝ 1024×1024 バイト）単位で。メモリマッピングの無効化は、非 Windows システム上において、リモートファイルが使用されている時に突然利用できなくなった場合の問題を回避するために使用できます。このオプションを 0（ゼロ）に設定すると、メモリマッピングは無効化されます。デフォルト：32 ビットプラットフォーム上では 50、そうでないなら 2048
searchpath[1]	（名前文字列のリスト）読み込みたいファイルの入ったディレクトリの相対パス名か絶対パス名（複数可）。この検索パスは複数回設定することができます。その場合、エントリは蓄積されて、設定された順に使用されます。空白キャラクタを含むディレクトリ名による問題を避けるために、エントリが 1 個だけであっても二重中括弧を用いることを推奨します。空文字列（すなわち {{}}）を指定すると、デフォルトエントリを含む既存の検索パスエントリがすべて削除されます。Windows の場合、searchpath はレジストリエントリで設定することも可能です。デフォルト：空
userlog	（名前文字列）ログ記録が有効にされている場合にログファイルへ書き込まれる任意の文字列。

1. オプションの値は複数回の呼び出しによって蓄積させることが可能です。

8.9 ログ記録

ログ記録機能を使うと、API 呼び出し群をトレースすることができます。そのログファイルの内容は、デバッグ目的に役立つ可能性があり、また、PDFlib GmbH サポートによって求められる場合があります。に、*PLOP_set_option()* を用いてログ記録記録を有効化するためのオプションを挙げます：

表 8.13 PLOP_set_option( ) に対するログ記録関連キー

キー	説明
logging	表 8.14 に従ったログ記録オプション群を持ったオプションリスト
userlog	ログファイルへコピーされる文字列

ログ記録オプション群は以下の方法で与えることができます：

- *PLOP_set_option()* の *logging* オプションに対するオプションリストとして。例：

```
plop.set_option("logging={filename={debug.log} remove}");
```

- *PLOPLOGGING* という環境変数内で。こうすると、いずれかの API メソッドへのいちばん最初の呼び出しからログ記録出力を有効化できます。
- （非サポート）次のレジストリキー内の中で：

```
HKLM\SOFTWARE\PDFlib\PLOP5\PLOPLOGGING
```

表 8.14 PLOP_set_option( ) に対する logging オプションに対するサブオプション

キー	説明
classes	（オプションリスト）各オプションがログ記録クラスを記述し、その照応する値がそのレベルを記述する、オプションリスト。レベル 0 はログ記録クラスを無効化し、正数はクラスを有効化します。レベルを上げると出力が詳細になります。クラスに対してレベルを指定しない場合は値 1 が用いられます（初期値：api=1）。 ***api*** すべての API 呼び出しを、その関数引数と戻り値とともにログ記録します。api=2 の場合、すべての API トレース行の頭にタイムスタンプが作成され、かつ、非推奨の関数とオプションにマークが付きます。 ***convert*** 文字列変換。 ***digsig*** 電子署名作成に関するログ詳細： ***1*** 基本情報 ***2*** 検証情報。OCSP と CRL の詳細。PKCS#11 ライブラリ・スロット・トークン情報 ***5*** 証明書の詳細 ***filesearch*** SearchPath か PVF からファイルを見つけようとする試みをすべてログ記録。 ***network*** ネットワーク活動に関する詳細をログ記録： ***1*** 一般的なネットワーク情報 ***2*** ネットワークヘッダ・統計 ***3*** 詳細なネットワークデータ ***resource*** Windows レジストリ・UPR 定義からリソースを見つけようとするすべての試みと、そのリソース検索の結果をログ記録。 ***user*** userlog オプションを用いて与えられる、ユーザー指定のログ出力。 ***warning*** 警告を、すなわち、無視することも内部的に修正することもできるエラー状況をすべてログ記録。warning=2 の場合、例外を発生させずに、PLOP_get_errmsg( ) から取得できるメッセージテキストを残す関数からのメッセージと、ファイルを開く試み（ファイルを searchpath で検索）のすべての失敗の理由もログ記録されます。
disable	（論理値）ログ記録出力を無効化。デフォルト：false
filename	（文字列）ログファイルの名前（stdout・stderr も受け付けられます）。既存の内容がある場合には、出力はその末尾に追加されます。このログファイル名は、PLOPLOGFILENAME という環境変数で与えることも可能です（その場合にはこのオプション filename は無視されます）。デフォルト：plop.log（Windows・macOS では / ディレクトリ内、Unix では /tmp 内）
flush	（論理値）true の場合、出力ごとにログファイルを閉じて、次の出力の際に再び開くことによって、出力が必ず実際に書き出されるようにします。これは、プログラムがクラッシュしてログファイルが途中で切れている場合の追跡に役立つ可能性がありますが、処理を非常に遅くします。false の場合、ログファイルは 1 回だけ開かれます。デフォルト：false
includepid	（論理値。MVS では不可）ログファイル名内にプロセス ID を含めます。複数のプロセスが同一のログファイル名を使用する場合にはこれを有効化するべきです。デフォルト：false
includetid	（論理値。MVS では不可）ログファイル名内にスレッド ID を含めます。同一プロセス内の複数のスレッドが同一のログファイル名を使用する場合にはこれを有効化するべきです。デフォルト：false
includeoid	（論理値。MVS では不可）ログファイル名内にオブジェクト ID を含めます。同一スレッド内の複数の PLOP オブジェクトが同一のログファイル名を使用する場合にはこれを有効化するべきです。デフォルト：false
remove	（論理値）true の場合、新規出力を書き込む前に既存のログファイルが削除されます。デフォルト：false

表 8.14 PLOP_set_option( ) に対する logging オプションに対するサブオプション

キー	説明
removeon-success	（論理値）例外が起こらない限り、生成されたログファイルを PLOP_delete( ) で除去します。これは、マルチスレッドのアプリケーションで時たま発生する問題や、散発的にのみ発生する問題を分析するために役立つ可能性があります。このオプションを includepid/includetid/includeoid と適切に組み合わせることを推奨します。
restore	（論理値）すべてのログ記録クラスレベルの状態（これと同じオプションリストで指定されているものを除く）を、いちばん最近に保存された状態へ復帰させます。
save	（論理値）すべてのログ記録クラスレベルの状態（これと同じオプションリストで指定されているものを除く）を保存します。7 個までの保存レベルを使えます。
stringlimit	（整数）テキスト文字列内のキャラクタ数の上限、あるいは 0 で無制限。デフォルト：0

8.10 pCOS 関数

PDF からオブジェクトデータを取得するための完全な pCOS 文法に対応しています。詳しい説明は pCOS パスリファレンスを参照してください。

C++ Java C# ***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, ...)***

数値型か**論理値**型の pCOS パスの値を得ます。

doc *PLOP_open_document*()* で取得した有効な文書ハンドル。

path 数値オブジェクトか論理値オブジェクトに対する完全な pCOS パス。

追加の引数群 （C 言語バインディングのみ）*key* 引数にプレースホルダがある場合、それに対応する任意の数の追加引数を与えることができます（***%s*** で文字列、***%d*** で整数。***%%*** を用いると 1 個のパーセント記号になります）。これらの引数を利用すれば、可変の数値や文字列値を含む複雑なパスをいちいち構成する手間が省けます。プレースホルダの数と型が、与える追加引数群に一致するようにするのは、クライアント側の役割です。

戻り値 pCOS パスで示されたオブジェクトの数値。論理値の場合、*true* なら 1 が返され、そうでないなら 0 が返されます。

C++ Java C# ***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, ...)***

名前・数値・文字列・論理値のいずれかの型の pCOS パスの値を得ます。

doc *PLOP_open_document*()* で取得した有効な文書ハンドル。

path 文字列・数値・名前・論理値のいずれかのオブジェクトに対する完全な pCOS パス。

追加の引数群 （C 言語バインディングのみ）*key* 引数にプレースホルダがある場合、それに対応する任意の数の追加引数を与えることができます（***%s*** で文字列、***%d*** で整数。***%%*** を用いると 1 個のパーセント記号になります）。これらの引数を利用すれば、可変の数値や文字列値を含む複雑なパスをいちいち構成する手間が省けます。プレースホルダの数と型が、与える追加引数群に一致するようにするのは、クライアント側の役割です。

戻り値 pCOS パスで示されたオブジェクトの値の文字列。論理値の場合、文字列 *true* か *false* が返されます。

詳細 pCOS がフルモードで動作していないとき、かつオブジェクトの型が**文字列**の場合には、この関数は例外を発生させます。ただし、*/Info/** オブジェクト群（文書情報キー）は制限 pCOS モードでも *nocopy=false* か *plainmetadata=true* なら取得することができ、また、*bookmarks[...]/Title* と、*pages[...]/annots[...]/* で始まるすべてのパスは、制限 pCOS モードでも *nocopy=false* なら取得できます。

この関数では、PDF 文書から得られる文字列はテキスト文字列であると前提しています。バイナリデータの入った文字列オブジェクトは、これでなく *PLOP_pcos_get_stream()* で取得するべきで、それならデータは一切変改されません。

バインディング C 言語バインディング：文字列は BOM なしの UTF-8 形式で返されます。返される文字列は、最大 10 エントリを持つリングバッファ内に格納されます。10 個を超える文字列がクエリされたときには、バッファは再利用されますので、10 個を超える文字列を同時に利用したい場合には、クライアント側でその文字列を複製しておく必要があります。たとえば、***printf()*** 文の引数ではこの関数を最大 10 回まで呼び出すことができます。同時に 10 個を超える文字列が使用されないならば、その戻り文字列は互いに独立であることが保証されているからです。

C++ 言語バインディング：文字列は、C++ ラッパのデフォルト ***wstring*** 構成における ***wstring*** として返されます。zSeries の ***string*** 互換モードでは、結果は BOM のない EBCDIC-UTF-8 形式で返されます。

C バインディング：返された文字列は、次にこの関数を呼び出すまでのあいだ使用できます。

Java・.NET：結果は Unicode 文字列として提供されます。

Perl・PHP・Python・Ruby 言語バインディング：結果は UTF-8 文字列として提供されます。

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, ...)

***stream*・*fstream*・文字列**のいずれかの型の pCOS パスの値を得ます。

doc *PLOP_open_document*()* で取得した有効な文書ハンドル。

length （C・C++ 言語バインディングのみ）返されるストリームデータの長さをバイト単位で受け入れる変数へのポインタ。

optlist 表 8.15 に従っていくつかの取得オプションを指定するオプションリスト。

path ストリームオブジェクトか文字列オブジェクトに対する完全な pCOS パス。

追加の引数群 （C 言語バインディングのみ）***key*** 引数にプレースホルダがある場合、それに対応する任意の数の追加引数を与えることができます（***%s*** で文字列、***%d*** で整数。***%%*** を用いると 1 個のパーセント記号になります）。これらの引数を利用すれば、可変の数値や文字列値を含む複雑なパスをいちいち構成する手間が省けます。プレースホルダの数と型が、与える追加引数群に一致するようにするのは、クライアント側の役割です。

戻り値 ストリームか文字列に入っている暗号化されていない状態のデータ。ストリームまたは文字列が空のとき、あるいは、暗号化されていない文書の中の暗号化された添付の内容がクエリされてその添付パスワードが与えられていないときは、返されるデータは空（C・C++ では NULL）になります。

オブジェクトが ***stream*** 型のときは、すべてのフィルタがストリームの内容から除去されます（すなわち、実際の生データが返されます）。オブジェクトが ***fstream*** 型か**文字列**型のときは、データは PDF ファイル内で見つかったそのままで返されますが、ただし例外として ASCII85・ASCII-Hex フィルタは除去されます。

詳細 pCOS がフルモードで動作していないとき、この関数は例外を発生させます。例外として、***/Root/Metadata*** オブジェクトは制限 pCOS モードでも ***nocopy=false*** か ***plainmetadata=true***

なら取得することができます。*path* が ***stream・fstream・文字列***型のオブジェクトを指していないときにも例外が発生します。

名前と違ってこの関数は、**文字列**型のオブジェクトを取得するためにも使えます。*PLOP_pcos_get_string()* の場合、オブジェクトをテキスト文字列として取り扱いますが、それとは違ってこの関数では、返すデータに一切の変改を加えません。バイナリ文字列データは PDF 内で用いられることは稀で、自動的に検出しようとしても確実ではありません。ですので、文字列オブジェクトをバイナリデータとして取得するかテキストとして取得するか、考えて適切な関数を選ぶのはユーザー側の役割です。

バインディング C・C++ 言語バインディング：返されたデータバッファは、次にこの関数を呼び出すまでのあいだ使用できます。

表 8.15 PLOP_pcos_get_stream( ) に対するオプション

オプション	説明
convert	（キーワード。非対応のフィルタで圧縮されているストリームに対しては無視されます）文字列またはストリームの内容が圧縮されるかどうかを制御（デフォルト：none）： ***none*** 内容をバイナリデータとして扱い、一切変換しません。 ***unicode*** 内容をテキストデータとして（すなわち PLOP_pcos_get_string( ) と全く同様に）扱い、Unicode に規格化します。Unicode 非対応の言語バインディングの場合、これはデータが BOM なしの UTF-8 形式に変換されることを意味します。 このオプションは、PDF 内でめったに使われないデータ型「テキストストリーム」（JavaScript 等のために使われます。ただし JavaScript の大多数はストリームオブジェクトでなく文字列オブジェクト内に格納されます）のために必要です。

8.11 Unicode 変換関数

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)

任意のエンコーディングの文字列を、さまざまな形式の Unicode 文字列へ変換します。

inputformat 入力文字列の解釈を指定する Unicode テキスト形式またはエンコーディング名：

- Unicode テキスト形式：***utf8***・***ebcdicutf8***（EBCDIC プラットフォーム上で）・***utf16***・***utf16le***・***utf16be***・***utf32***
- すべての内部的に知られている 8 ビットエンコーディングと、ホストシステム上で利用可能なエンコーディングと、日中韓エンコーディング ***cp932***・***cp936***・***cp949***・***cp950***
- キーワード ***auto*** は次の動作を指定します：入力文字列に UTF-8 または UTF-16 BOM がある場合はそれを用いて適切な形式が決定され、ない場合はカレントのシステムコードページであると見なされます。

input Unicode へ変換したい文字列

inputlen （C 言語バインディングのみ）入力文字列の長さをバイト単位で。***inputlen = 0*** の場合には、ヌル終端文字列を与える必要があります。

outputlen （C 言語バインディングのみ）返される文字列の長さ（バイト単位で）が格納されるメモリ位置への C スタイルのポインタ。

optlist 入力の解釈と Unicode 変換のためのオプション群を指定したオプションリスト：

- 表 8.16 に従った入力フィルタオプション群：***charref***・***escapesequence***
- 表 8.16 に従った Unicode 変換オプション群：***bom***・***errorpolicy***・***inflate***・***outputformat***

戻り値 指定された引数とオプションに従って入力文字列から生成された Unicode 文字列。入力文字列が、指定された入力形式に準拠していないとき（無効な UTF-8 文字列など）は、***errorpolicy=return*** の場合には空の出力文字列が返され、***errorpolicy=exception*** の場合には例外が発生します。

詳細 この関数は、汎用の Unicode 文字列変換に役立つでしょう。これは、適切な Unicode コンバータを提供していない環境で作業をするユーザーの便宜のために提供されています。

バインディング C バインディング：返される文字列は、最大 10 エントリを持つリングバッファ内に格納されます。10 個を超える文字列が変換されたときには、バッファは再利用されますので、10 個を超える文字列を同時に利用したい場合には、クライアント側でその文字列を複製しておく必要があります。たとえば、***printf( )*** 文の引数ではこの関数を最大 10 回まで呼び出すことができます。同時に 10 個を超える文字列が使用されないならば、その戻り文字列は互いに独立であることが保証されているからです。

表 8.16 PLOP_convert_to_unicode( ) に対するオプション

オプション	説明
bom	（キーワード。outputformat=utf32 の場合には無視されます。.NET・Java・Objective-C・Python では none のみ許されます）バイト順序マーク（BOM）を出力文字列に加えるかどうかの方針。使えるキーワード（デフォルト：none）： ***add*** BOM を加えます。 ***keep*** 入力文字列に BOM があるなら BOM を加えます。 ***none*** BOM を加えません。 ***optimize*** outputformat=utf8 または ebcdicutf8 かつ出力文字列が範囲＜U+007F. のキャラクタのみ含む場合以外には BOM を加えます。
charref	（論理値）true の場合、数値・文字実体参照とグリフ名参照の置き換えを有効にします。デフォルト：false
errorpolicy	（キーワード）変換エラーの場合の動作（デフォルト：exception）： ***return*** 文字参照が解決できないときに代替キャラクタが使用されます。変換エラーの場合に空文字列が返されます。 ***exception*** 変換エラーの場合に例外が発生します。
escape-sequence	（論理値）true の場合、文字列内のエスケープシーケンスの置き換えを有効にします。デフォルト：false
inflate	（論理値。inputformat=utf8 の場合のみ。outputformat=utf8 の場合には無視されます）true の場合、無効な UTF-8 入力文字列が来ても例外が発生せず、指定された出力形式のインフレートされたバイト文字列が生成されます。これはデバッグのために役立つでしょう。デフォルト：false
output-format	（キーワード）生成される文字列の Unicode テキスト形式：utf8・ebcdicutf8（EBCDIC プラットフォーム上で）・utf16・utf16le・utf16be・utf32。空文字列は utf16 と同等です。デフォルト：utf16 Unicode 対応言語バインディング：出力形式は utf16 を強制されます。 C++ 言語バインディング：次の出力形式のみ許されます：utf8・utf16・utf32。

A 証明書を用いる作業

この付章では、証明書と PLOP または PLOP DS を用いて作業を行う際に役立つ可能性のあるさらなる情報を提供します。

証明書の内容を表示　証明書の、あるいは PKCS#12 形式のデジタル ID の内容を表示するには、以下の Windows コマンドを用います：

```
certutil -dump -p demo demo_signer_rsa_2048.p12
```

OpenSSL を用いて、PEM エンコーディングの証明書の内容を表示：

```
openssl x509 -inform PEM -in demo_recipient_1.pem -noout -text
```

OpenSSL を用いてデジタル ID から公開鍵を抽出して証明書を作成　もし、PKCS#12 形式のデジタル ID（公開鍵と秘密鍵を有しています）を持っており、照応する、他の人が文書を暗号化するための PEM エンコーディングの証明書（公開鍵のみを有しています）が必要な場合は、以下のコマンドを使用できます。パスワードを聞いてきます：

```
openssl pkcs12 -in demo_recipient_1.p12 -clcerts -nokeys -out ↵
        demo_signer_rsa_2048.pem
```

OpenSSL を用いて証明書を PEM へ変換　署名オプション *certfile・rootcertfile* と、ネットワークオプションリスト内のサブオプション *sslcertfile* は、テキストベースの PEM エンコーディングの証明書のみを受け付けます。EBCDIC プラットフォームでは PEM 証明書は EBCDIC で符号化されている必要があります。

以下の OpenSSL コマンドを使用することによって、バイナリ DER エンコーディングの証明書を、必要な、テキストベースの PEM エンコーディングへ変換することができます：

```
openssl x509 -inform DER -outform PEM -in PDFlibDemoCA_G3.crt -out PDFlibDemoCA_G3.pem
```

.cer ファイルと *.crt* ファイルは、DER エンコーディングを使用している場合もありますし、PEM エンコーディングを使用している場合もあることに留意してください。ファイル名の拡張子では区別がつきませんので、テキストエディタで形式をチェックできます :DER エンコーディングの中身はバイナリデータであり、一方、PEM エンコーディングは以下の行で挟まれた Base-64 符号化されたデータを有するテキストベースの形式です：

```
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

証明書・CRL ファイルの命名規則　署名オプション *crldir・rootcertdir* と、ネットワークオプションリスト内のサブオプション *sslcertdir* は、証明書と CRL が検索されるディレクトリの名前を受け付けます。これらのファイルは、テキストベースの PEM エンコーディングで保管されている必要があり、かつ、OpenSSL 1.0.0（およびそれ以降）のファイル名ハッシュ化規則に従って命名されている必要があります。

OpenSSL コマンド *c_rehash* は、PEM エンコーディングの証明書群か CRL 群を含んだ 1 個ないし複数のディレクトリに対して、必要なハッシュ化されたファイル名を持ったシンボリックリンク群を作成します：

```
c_rehash .
```

ハッシュ化されたファイル名を手作業で作成したい場合は、以下の手順で行います：

- 以下のいずれかのようなOpenSSLコマンドを用いて、個別の証明書またはCRLファイルに対してファイル名ハッシュを作成：

```
# PEMエンコーディングの証明書に対してハッシュ化されたファイル名を作成
openssl x509 -hash -noout -in PDFlibDemoCA_G3.pem

# DERエンコーディングの証明書に対してハッシュ化されたファイル名を作成
openssl x509 -hash -noout -inform DER -in PDFlibDemoCA_G3.crt

# PEMエンコーディングのCRLに対してハッシュ化されたファイル名を作成
openssl crl -hash -noout -in PDFlibDemoCA_G3.crl.pem

# DERエンコーディングのCRLに対してハッシュ化されたファイル名を作成
openssl crl -hash -noout -inform DER -in PDFlibDemoCA_G3.crl
```

- 末尾に「.」（ピリオド）キャラクタを付加。CRL の場合はさらにキャラクタ「r」も付加。
- 10 進数値 0（ゼロ）を付加。ディレクトリ内で、ハッシュ化されたファイル名の衝突がある場合は、この数値を 1 ずつ増やしていきます。

注記 zSeries 用 OpenSSL では、バグのため、互換なファイル名ハッシュを算出することができません。

オブジェクト識別子（OID）の構文　オプションのサブオプションと、オプションのサブオプションは、ポリシーを指定するオブジェクト識別子（OID）を受け付けます。OID は 10 進数値の列で構成され、それぞれの数値は空白またはピリオドキャラクタ「.」によって区切られます。例：

```
2.16.840.1.101.3.2.1.48.9
```

B PDFlib を PLOP DS と結合

PLOP DS は、PDF 文書を動的に生成してそれに署名するために、PDFlib と容易に相互動作するよう設計されています。この付章では、この 2 つの製品を結合する方法を説明します。

ファイルベースでの結合　ファイルベース方式は、非常に大きな PDF 文書を扱う場合や、PDFlib/PLOP DS 結合の総メモリ要求を下げる必要がある場合に推奨します。単に、適切な PDFlib ルーチンで PDF ファイルをディスク上に生成した後、その生成された文書を *PLOP_open_ document()* で処理します。

文書をメモリ内に作成して電子的に署名　メモリベース方式は比較的速いですが、メモリを比較的多く必要とします。非常に大きな文書を扱う場合を除いて、これは Web アプリケーションで動的な PDF 生成や署名を行う場合に推奨します：

- PDFlib で PDF ファイルをディスク上に生成するのではなく、*PDF_begin_document()* に空のファイル名を与えることによってインコア PDF 生成を利用します。
- 生成されたPDFデータを、*PDF_end_document()*の後に*PDF_get_buffer()*を呼び出すことによって取り出します。
- このPDFデータに基づいて、*PLOP_create_pvf()*を呼び出すことによってPLOP内に仮想ファイルを作成します。
- この PVF ファイルの名前を、*PLOP_open_document()* を用いて PLOP DS へ渡します。

すべての PLOP パッケージに入っている *hellosign* プログラミングサンプルでは、PDFlib を使って動的に PDF 文書を生成し、それを PLOP にメモリ内で渡して電子署名を適用する方法を示しています。

署名視覚化文書を動的に作成　署名視覚化のために用いる文書を、PDFlib を用いて動的に作成することもできます（7.3.1 節「グラフィックかロゴを用いて署名を視覚化」（105 ページ）参照）。これは、視覚化文書の中に現在日時等可変のテキストまたは画像構成要素を含める必要がある場合に役立つでしょう。

dynamicsign プログラミングサンプルは、PDFlib を用いて PDF 視覚化文書を動的に作成し、それを署名作成処理で使用するために PLOP DS に渡す方法を演示しています。

PDFlib 9.2 以下を用いて作成されたフォームフィールド　Acrobat は、フォームフィールド群を内容として持つ文書を開く際、必要に応じて、その PDF 文書の中のフィールド群の視覚表現を自動的に保持します。PDFlib 9.2 以下では、この動作に依存しており、いわゆる体裁ストリームを生成しません。しかし、Acrobat の自動体裁生成は、文書を開いた直後に文書を変更しますので、そのような文書は電子署名には適していません。

ですので、PLOP DS はデフォルトで（更新モードで署名する場合）そのような文書を拒否します。書き換えモードでは、すなわち *update=false* では、そのような文書には、*sacrifice={fields}* オプションを与えることによって署名できます。ただし、このオプションを使うと、入力文書内のフォームフィールド群は、署名が行われた出力の中では存在しなくなります。

PDFlib 9.3 以上を用いて生成されたフォームフィールド文書については制約は何もありません。

PDFlib を用いて作成された注釈　もし PDFlib を使用して注釈のある文書を生成し、その後に PLOP DS を用いてその文書に署名を行うと、Actobat はその署名を有効と表示しま

す。しかし、Acrobat が「Annotations Modified」（注釈が変更されています）と表示することもあります（その署名を検証した後にのみ、ときどき）。その原因は、Acrobat は、文書を開く際に、注釈が名前項目を持っていないと、それを黙って追加するからです。また、Acrobat は「体裁ストリーム」も追加します。これは、体裁の視覚表現です。これらの変更が、文書内の変更に関する警告メッセージを引き起こすのです。このまぎらわしいメッセージが出ないようにするには、PDFlib を用いて入力文書を生成する際に、以下を行います：

- 注釈を作成する際にその注釈名を与える。これを行うには、PDFlib API メソッド ***PDF_create_annotation()*** の ***name*** オプションを用います。
- 体裁ストリーム（すなわちテンプレート）を与える。PDFlib API メソッド ***PDF_create_annotation()*** の ***template*** オプションとサブオプション ***normal*** を用います。

C PLOP ライブラリクイックリファレンス

以下の表は、すべての PLOP API メソッドの概観です。

一般メソッド

関数プロトタイプ	ページ
（C のみ）*PLOP *PLOP_new(void)*	140
void delete( )	140
void create_pvf(String filename, byte[ ] data, String optlist)	140
int delete_pvf(String filename)	140
double info_pvf(String filename, String keyword)	142

文書入力・出力

関数プロトタイプ	ページ
int open_document(String filename, String optlist)	143
（C のみ）***int PLOP_open_document_callback(PLOP *plop, void *opaque, plop_off_t filesize, size_t (*readproc)(void *opaque, void *buffer, size_t size), int (*seekproc)(void *opaque, plop_off_t offset), const char *optlist)***	145
int create_document(String filename, String optlist)	147
void close_document(int doc, String optlist)	146
byte[ ] get_buffer( )	151
int prepare_signature(String optlist)	154

エラー処理

関数プロトタイプ	ページ
int get_errnum( )	166
String get_errmsg( )	166
String get_apiname( )	166

グローバルオプション

関数プロトタイプ	ページ
void set_option(String optlist)	168

pCOS メソッド

関数プロトタイプ	ページ
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

Unicode 変換

関数プロトタイプ	ページ
string convert_to_unicode(string inputformat, byte[ ] input, string optlist)	*176*

D 変更履歴

このマニュアルの変更履歴

日付	変更
2020 年 06 月 14 日	► PLOP 5.4・PLOP DS 5.4 に関する変更
2018 年 08 月 28 日	► PLOP 5.3・PLOP DS 5.3 に関する変更
2017 年 12 月 11 日	► PLOP 5.2p2 に関する若干の変更：Acrobat の EC 暗号化対応につき明確化。注釈を含む PDF 作成文書への署名について
2017 年 05 月 23 日	► PLOP 5.2・PLOP DS 5.2 に関する若干の変更
2016 年 10 月 27 日	► PLOP・PLOP DS 5.1r1 の証明書セキュリティと若干の新機能に関する変更
2016 年 05 月 13 日	► PLOP 5.1・PLOP DS 5.1 に関する変更
2015 年 03 月 27 日	► PLOP・PLOP DS 5.0 r1 に関する若干の変更
2015 年 02 月 27 日	► PLOP 5.0・PLOP DS 5.0 に関する変更
2011 年 03 月 04 日	► PLOP 4.1・PLOP DS 4.1 のためのメジャーオーバーホール
2008 年 12 月 05 日	► PLOP 4.0・PLOP DS 4.0 の XMP・PVF・PKCS#11（スマートカード）対応に関する更新
2007 年 07 月 15 日	► PLOP 3.0 と PLOP DS 3.0 に関する更新
2004 年 09 月 27 日	► PLOP 2.1 に関する更新
2003 年 12 月 01 日	► 新しいメジャーリリース PLOP 2.0 に関する更新
2002 年 11 月 23 日	► Perl 用 PSP バインディングの記述を追加
2002 年 11 月 07 日	► ILE-RPG での PSP の利用に関する節を追加
2002 年 10 月 22 日	► PSP 1.0.1 に関する若干の変更
2002 年 09 月 17 日	► PSP 1.0.0 に関する第一版

索引

記号

A

B

C

D

E

G

H

I

J

L

M

N

PDFlib GmbH

PDFlib GmbH
Franziska-Bilek-Weg 9
80339 München, Germany
www.pdflib.com
電話 +49・89・452 33 84-0

ライセンスに関するお問い合わせ
sales@pdflib.com

サポート
support@pdflib.com （お使いのライセンス番号をお書きください）