ラング・エッジ PDF 署名ライブラリ

LE:PAdES:Lib マニュアル

２０２４年2月２０日版

□ 目次

○ 履歴

バージョン	日付	主な修正内容
Ver 1.09.B1	2024/02/20	署名アルゴリズムとして ECDSA（楕円曲線暗号）へ対応。 「6.1. 検証結果 XML 仕様」への署名アルゴリズム情報の追加
Ver 1.08.R3	2024/01/22	「9. PDF 解析結果 XML」への署名オプション情報出力の追加
Ver 1.08.R1	2023/08/30	LE:PKI:Lib（PKI 要素と利用）のマニュアルの分離に伴う「4. PKI 要素と署名方法」の簡素化 OpenSSL 3.1.1 対応に伴う「1.9. 外部利用ライブラリ」の修正 「1.1. 全体構成と長期署名」の修正
Ver 1.07.R6a	2023/02/13	OpenSSL 1.1.1t 対応に伴うファイル構成の変更に伴う修正。 「1.5. 動作に必要なファイル」の修正。 「1.6. Windows 環境のインストールとソースビルド」の修正。 「1.9. 外部利用ライブラリ」の修正。 「2.5. .NET API 利用の場合」の修正。
Ver 1.07.R5 Ver 1.07.R4	2022/09/01	「7.4. 署名サーバ実装例」に LpaCmd の実装例を追加。 「A.5. LePAdES エラーコード」に PDA_ERR_ALREADY_SIGNED 追加
Ver 1.07.R3	2022/02/14	「3.8. 長期検証（LTV）対応の判定」の追加。 「3.9. 署名操作時の XMP 更新」の追加。 「付録 A. エラーコード」の更新。
Ver 1.07.R2	2021/07/07	4 章を見直して内容を更新した。特に CAPI の IC カード利用については V1.07.R2 の新機能としてスペック指定が可能となったので詳細を追加している。また HSM 利用（仮署名）についての説明を追加した。
Ver 1.07.R1	2021/01/15	「1.9. 利用外部ライブラリ」の更新。 「3.1. PDF 暗号化と電子署名」の更新。 「7. クライアント署名」の更新（V2.1 対応）
Ver 1.06.R2	2019/05/30	「8.3. 添付ファイル」と「8.4. PDF 情報」の追加。
Ver 1.06.R1	2018/11/26	「3.4. 署名外観」へ透過 PNG 画像による透過署名外観の追加。 「3.7. Adobe 仕様の失効情報（V1.06.R1 以降)」の追加。 「4.9. 失効情報取得の高度な指定（v1.06.R1 以降)」の追加。 「4.10. HSM（ハードウェア・セキュリティ・モジュール)」の追加。 「6.2. 署名解析 XML 仕様（PdaParseXml)」の廃止（9.2. に変更)。 「7. クライアント署名」の V2 バージョンアップ対応 「9. PDF 解析 XML」の追加と「10. ネットワーク機能」の章番号更新。
Ver 1.05.R1	2016/12/21	ビルド環境の変更。Windows 版の VS2012/2013/2015 対応と、Linux 版デフォルトを 64bit に変更等。 「9.1. HTTP 通信のプロキシ設定」の Windows 対応の更新。 「4.8. 証明書検証サーバ（CVS）の利用：GPKI/LGPKI 等」の追加。
Ver 1.04.R3	2016/03/30	「2.5. .NET API 利用の場合」を追加 その他 .NET の API 追加に関する記述を各所に追加
Ver 1.04.R2	2015/06/04	「4.7. PKCS#11 対応 IC カードによる署名の利用」を追加
Ver 1.04.R1	2015/01/13	「8. PDF 操作」と「9. ネットワーク機能」を追加
Ver 1.03.R2	2014/08/25	「3.3. 署名辞書」にロック署名を追加 「5.2. 署名辞書 XML 仕様（PdaSign)」にロック署名を追加
Ver 1.03.R1	2014/05/02	「1.3. 評価版」を追加 「1.6. Windows 環境のインストールとソースビルド」VS2010 用の修正 「1.10. フォルダ構成」他、フォルダ構成変更に伴い関連各所を修正 「3.1. PDF の暗号化と電子署名」開くパスワードのサポートを追加 「4.1. 証明書」に動作確認済みの証明書一覧を追加 「4.2. タイムスタンプ」のサービス整理と Amano サポートに伴う修正 「付録 C. FAQ（よくある質問)」を追加

Ver 1.02.R2	2013/10/21	「3.6. 長期署名の運用」を追加
Ver 1.02.R1	2013/09/12	「1.8. 利用外部ライブラリ」にコピーライト表示を追加 「2.5. LpaCmd コマンドの利用例」の「4」長期保管」の修正 「4.6. LangEdge 試験用 PKI 環境」の追加 「付録 B. 制限事項」の修正、「付録 C. コピーライト表示」の追加
Ver 1.02.B1	2013/08/09	「1.6. Linux 環境のインストールとソースビルド」を修正 「1.8. 外部利用ライブラリ」「1.9. フォルダ構成」を修正 「2.3. Linux C++利用の場合」「2.4. Java 利用の場合」を修正 「1.10. 文字コードと LeString クラス」と「3.5. 署名検証」を追加
Ver 1.01.R3	2013/07/10	「4.5. Windows 証明書ストア」を追加 「2.4. Java 利用の場合」に finalize() の注意を追加 「付録 A. エラーコード」へのエラー追加 「付録 B. 制限事項」の修正
Ver 1.01.R1	2013/06/13	LE:PAdES:Lib マニュアルの第 1 版

１．概要

PAdES（ETSI TS 102 778・EN 319 142-1）とは長期保管に対応した PDF 電子署名の仕様名称であり、PDF 仕様の ISO 32000-2:2020 と PAdES プロファイル ISO 14533-3:2017 にて定義されている。本書はラング・エッジの「PDF 署名基本ライブラリ LE:PAdES-Basic:Lib（以後 LE:PAdES-Basic:Lib)」と「PDF 長期署名ライブラリ LE:PAdES:Lib（以後 LE:PAdES:Lib)」の製品マニュアルである。どちらも基本的な機能は同じであるが、簡単に言えば LE:PAdES:Lib がフル機能版であり、LE:PAdES-Basic:Lib がサブセット版（ライト版）である。LE:PAdES-Basic:Lib では利用できる API が一部制限（長期保管関連）されている。「LE:PAdES-Basic:Lib＋長期署名（長期保管）＝LE:PAdES:Lib」と言える。本書中では共通の機能に関しては LE:PAdES:Lib と記述する。

名称	概要
PDF 署名基本ライブラリ **LE:PAdES-Basic:Lib** **（サブセット版）**	長期署名に必要な検証情報埋め込みに非対応、短中期的な署名。 PDF 署名の PAdES 仕様のうち Part4 PAdES LTV の DSS/VRI 以外の機能をサポート。なお DSS/VRI に関しても検証（長期署名検証）は可能。
PDF 長期署名ライブラリ **LE:PAdES:Lib** **（フル機能版）**	PDF 長期署名にフル対応（LE:PAdES-Basic:Lib 機能を全て含む） PDF 署名の PAdES 全仕様の機能をサポート。長期保管に対応可能。 ※ 差額により LE:PAdES-Basic:Lib からのアップグレードが可能。

ラング・エッジ PDF 署名製品

PAdES 規格		**概要**	**PDF 長期署名 LE:PAdES:Lib**	**PDF 基本署名 LE:PAdES-Basic:Lib**
PAdES-Basic		PKCS#7 を使った従来型 PDF 電子署名 Adobe Reader 6 以降で検証可能	○	○
PAdES-Enhanced		CAdES を使った新しい PDF 電子（長期）署名 Adobe Reader-X 以降で検証可能	○	○ **(Ver1.05 より)**
PAdES-LTV	**DocTimestamp (RFC 3161)**	タイムスタンプのみ付与 （長期保管にも利用）	○	○
	DSS/VRI	検証情報の埋め込み （長期保管用）	○	**△（検証のみ） [長期保管不可]**

ラング・エッジ PDF 署名製品比較

※ Ver1.05 より PDF 基本署名ライブラリ LE:PAdES-Basic:Lib でも PAdES-Enhanced が使えるようになった。LpaCmd のデフォルトも –type cades となる。PAdES-Basic の PKCS#7 形式の新規署名は ISO 14533-3 では非推奨となっている。

※ LE:PAdES:Lib では PAdES 仕様のうち PAdES for XML Content（PDF に添付された XML ドキュメントまたは XFA:XML Forms Architecture として埋め込まれた XML フォームの長期署名仕様）は日本国内ではほぼ利用されていない為に対応しない。必要な場合には別途相談を。

※ Ver1.09 より RSA 署名アルゴリズム以外に ECDSA 署名アルゴリズムに対応した。

LE:PAdES-Basic:Lib と LE:PAdES:Lib は利用時に区別がつきにくいが API としては LePAdES::isFull() で、コマンドの –full 引数にて確認が可能。

LE:PAdES-Basic:Lib 戻り値：0	> LpaCmd -full LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) LangEdge **PAdES-Basic** Command. Copyright I 2012-20XX LangEdge,Inc. all rights reserved. >
Le:PAdES:Lib 戻り値：1	> LpaCmd -full LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) LangEdge **PAdES** Command. CopyrigI(c) 2012-20XX LangEdge,Inc. all rights reserved. >

ラング・エッジの既存製品である「XML 長期署名ライブラリ LE:XAdES:Lib」と同じく、LE:PAdES-Basic:Lib と LE:PAdES:Lib も、保守目的で利用可能な全てのソースコードとプロジェクトのファイルが付属する。ただしライセンスはオープンソースライセンスでは無く、商用ライセンスでの提供である。なお保守目的の場合には修正が可能。

※ Ver1.08.R1 の仕様変更に関する注意点

LE:PKI:Lib の Ver1.08.R1（2023 年 8 月リリース）では大きな仕様変更が 2 カ所あったので注意が必要となる。1 つは Windows/Linux 共通で検証時の CRL 優先から OCSP 優先への切り替えであり、もう 1 つは Windows のみであるが HTTP/HTTPS 通信が WinInet から標準 WinHTTP への切り替えである。どちらもデフォルト設定の変更であり、別途オプション指定により従来通りの動作も可能となっている。

仕様変更１：検証が CRL 優先から OCSP 優先になった

対象環境	Windows 版と Linux 版の両方で仕様変更
影響	証明書が CRL と OCSP の両方に対応している場合に従来は CRL を使っていたが変更後は OCSP を使うことになる。OCSP への署名証明書（OCSP レスポンダ証明書）の有効期間が短く、CRL の有効期間が長い場合には CRL 優先の指定をした方が良い場合がある。
変更理由	OCSP は通常リアルタイムに失効確認ができるので猶予期間が不要であることから標準を OCSP 優先への切り替えた。
従来仕様指定	検証フラグとして LPK_VERIFY_FLAG の LPKV_PRIOR_CRL を指定することで従来通り CRL 優先の検証が可能となる。
詳細	LE:PKI:Lib マニュアル（LePKI-manual.pdf）の次の章を参照。 ３．１１．失効情報取得の高度な指定

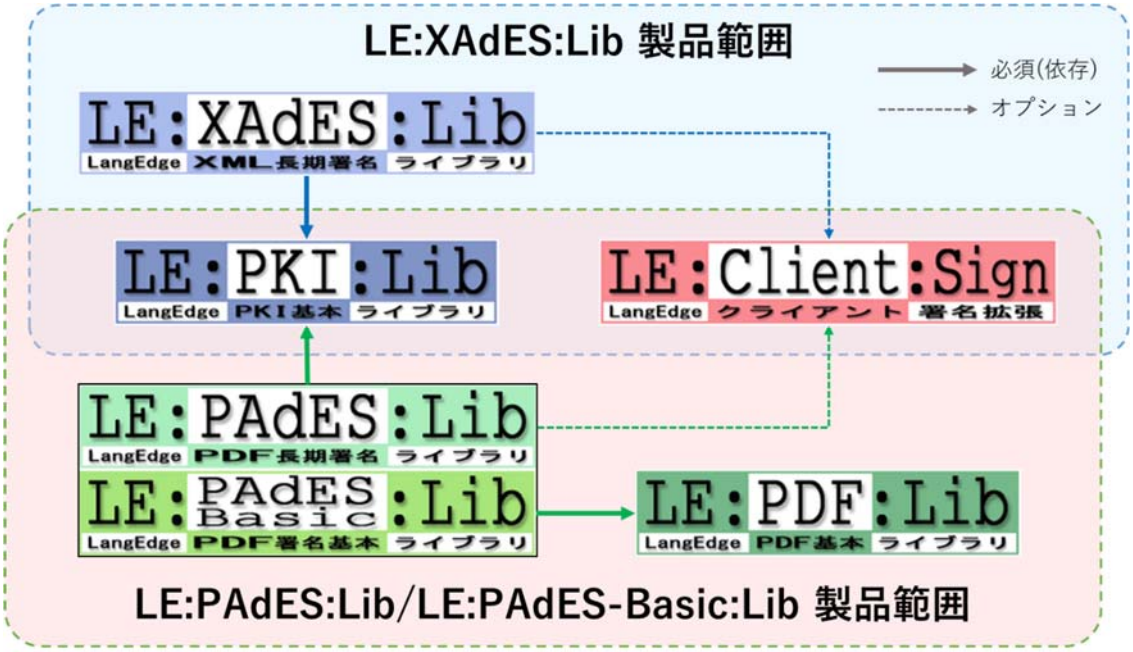
仕様変更２：HTTP/HTTPS 通信モジュールが WinInet から WinHTTP になった

対象環境	Windows 版のみの仕様変更（Linux 版は従来通りの仕様）
影響	プロキシ設定として WinHTTP 用に変更する必要がある可能性がある。ただし WinHTTP は Windows Update にも使われる通信モジュールであり、通常であればプロキシ等の設定は済んでいるはず。
変更理由	WinInet は IE の通信モジュールであり、IE 廃止に伴い将来的に対応されなくなる可能性がある為に Windows Update にも使われている WinHTTP に変更した。なお WinInet が廃止されると言う情報はまだ無い。
従来仕様指定	LePKI.setHttp(), LpkTimestamp.setHttp(), LpkUtil の getCrl()/getOcsp()/getOcsp2()/getCvs() の引数 http に LPK_HTTP_TYPE の LPK_HTTP_WININET を指定することで従来通り WinInet を使った通信が可能となる。
詳細	LE:PKI:Lib マニュアル（LePKI-manual.pdf）の次の章を参照。 ４．１．Windows 版：通信方式の指定

１．１．全体構成と長期署名

PDF 長期署名ライブラリ LE:PAdES:Lib は、大きく分けて PDF 操作と PDF 電子署名を行う LE:PAdES:Lib/LE:PAdES-Basic:Lib（内部に PDF 操作の LE:PDF:Lib を含む）と、暗号と PKI の操作を行う LE:PKI:Lib から構成される。またオプション製品としてサーバー上の LE:PAdES:Lib と連携してクライアント（Windows）上のブラウザ（Edge/Chrome）と連携して署名鍵/証明書を利用するクライアント署名拡張 LE:Clinet:Sign の利用も可能となっている。

ラング・エッジの電子署名製品としては XML 長期署名ライブラリ LE:XAdES:Lib も提供されている。LE:XAdES:Lib V3（Ver3）以降は、LE:PKI:Lib と LE:Client:Sign は共通して利用される。

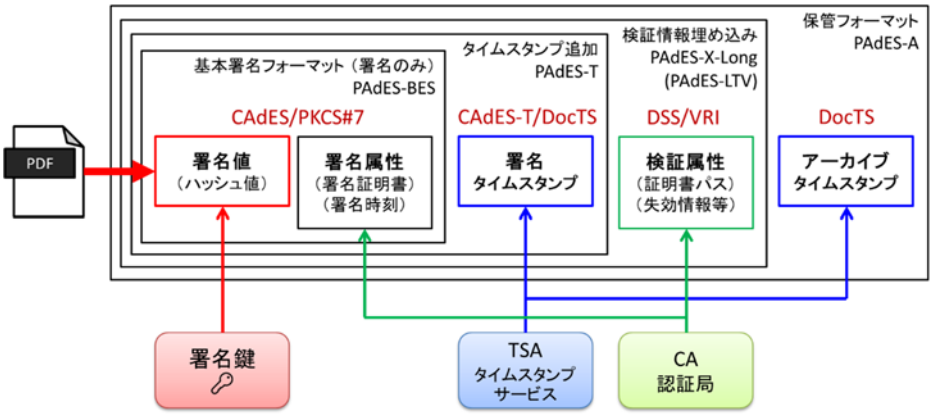


機能と全体の構成

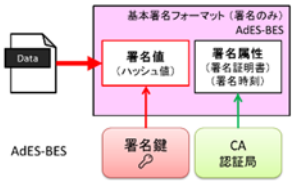
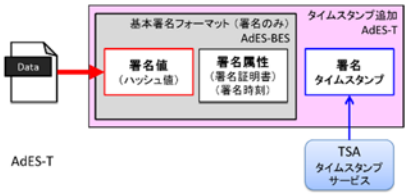
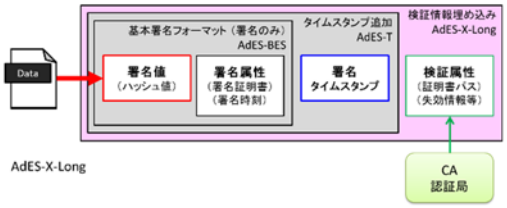
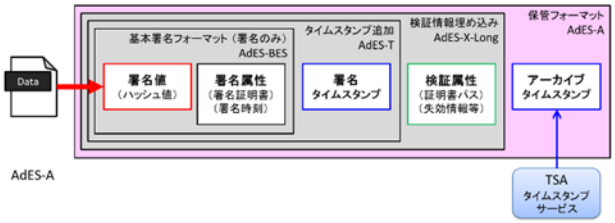
ライブラリ名		ドキュメント（マニュアル）	内容
LE:PAdES:Lib or LE:PAdES-Basic:Lib	LE:PDF:Lib	LePAdES-manual.pdf（本書）	PDF 署名の付与手順や検証の方法やオプション設定等の説明 ※　LE:PDF:Lib のマニュアルは無い。
LE:PKI:Lib（単体利用可能） ※　タイムスタンプ取得のみ等であれば LE:PKI:Lib だけで可能		LePKI-manual.pdf	署名方法や、証明書検証やタイムスタンプ利用の方法やオプション等の説明
LE:Client:Sign（単体利用不可） ※　利用には LE:PAdES:Lib または LE:XAdES:Lib が必要		LeClientSign2-manual.pdf	ブラウザ（Edge/Chrome）への署名機能の JavaScript 組み込み方法等の説明

機能とドキュメント構成

長期署名フォーマットは、署名データ＋タイムスタンプトークン＋検証情報（証明書＋失効情報）で構成される。これらを順に追加して行くことで生成して行くことになる。



一般的な長期署名構造

手順	構造		説明
Step.1 PAdES-BES 署名付与 API: addEnhanced() (addBasic は非推奨)		Step.1&2 PAdES-T 署名付与＋ 署名タイムスタンプ （SigTS）の付与 API: setSignTimestamp() + addEnhanced()	署名対象（PDF）を対象として署名証明書や署名時刻を付け署名データ（CAdES/ PKCS7）を生成する。
Step.2 PAdES-T 署名タイムスタンプ （DocTS） API: addTimestamp()			署名時刻を保証するタイムスタンプを追加する。 CAdES にタイムスタンプを埋め込む SigTS と、別途追加する DocTS がある
Step.3 PAdES-X-Long 検証情報埋め込み API: addLTV() 注： LE:PAdES-Basic:Lib では利用不可	 **※ 全証明書の検証情報が PDF 内に埋め込まれている状態**		署名証明書と TSA 証明書を検証し、認証パス証明書群と失効情報群を DSS/VRI 辞書に埋め込んで追加する。 **※ Adobe の LTV 仕様**
Step.4 PAdES-A アーカイブ タイムスタンプ API: addTimestamp()			全体を保護する為のアーカイブタイムスタンプを追加する。
Step.5 延長	---		Step.3 と 4 を繰り返す。

長期署名の作成手順

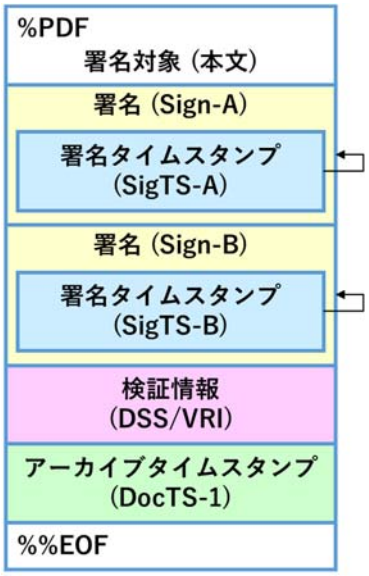
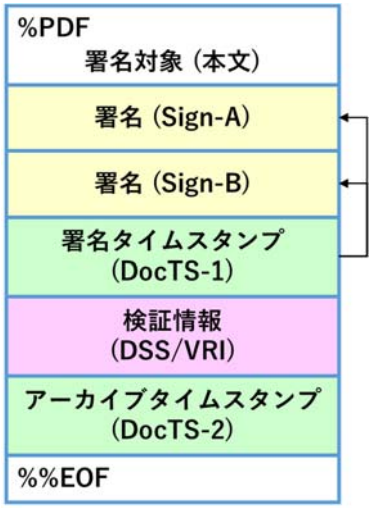
PDF 署名/PAdES では他の署名方式（XML 署名/XAdES や CMS 署名/CAdES）と異なる点がある。大きくは以下の 2 点となる。

差異 1 ：複数署名はシリアル署名として必ず増分更新により追加されて行く（並列署名不可）。
差異 2 ：タイムスタンプのみ（署名無し）の仕様が PAdES-DT として認められている。

差異のうち 2 つ目のタイムスタンプのみの PAdES-DT 仕様が認められていることは利点でありあまり問題にはならない。しかしながら 1 つ目の必ずシリアル署名（増分更新）になると言う点は複数の署名を付与したい場合に問題となる。追加されたタイムスタンプが最初の署名ではアーカイブタイムスタンプとなるのに次の署名では署名タイムスタンプとなるような例がある。これは PAdES では各署名を起点としてそれぞれ長期署名の構造を見ることになっている為に生じる問題である。詳しくは JNSA 電子署名 WG の「デジタル署名検証ガイドライン」の「付属書 B (参考): PAdES 関連情報」に解説がある。なお同じ内容は ISO 14533-3 にも記述されている。

JNSA：デジタル署名検証ガイドライン
https://www.jnsa.org/result/e-signature/2021/

ここでは電子契約のような複数署名を長期署名化する場合に推奨される構造を簡単に説明する。推奨 1 はアーカイブタイムスタンプのみ共有する構造で、推奨 2 は署名タイムスタンプ以降を共有する構造である。推奨 2 の方法ではタイムスタンプ数は削減できるが署名個々の署名時刻は保証されない。

推奨	1 ：アーカイブタイムスタンプのみ共有	2 ：署名タイムスタンプ以降を共有
構造図		
解説	必要となるタイムスタンプ数は増えるが最も推奨される構造。署名タイムスタンプは署名データ（CAdES）に埋め込む方式を推奨。	署名時刻を 1 つの署名タイムスタンプの時刻とする構造。署名タイムスタンプはドキュメントタイムスタンプとして追加する。

１．２．API 環境

LE:PAdES:Lib では C++で開発されており、インターフェイスとして C++ / Java / .NET / コマンドの 4 種類が提供される。Java 環境に関しては JNI を使っている為にピュア Java では無い、また.NET 環境に関しても C++/CLI を使っている為にピュアなマネージコードでは無いので注意が必要である。

API 環境	C++	Java	.NET	コマンド
補足	.DLL/.so 呼び出し	JNI 経由により C++呼び出し	ラッパ経由により C++呼び出し	コマンド引数にてオプション指定
利用例	アプリ組み込み等	Web サービスへの組み込み等	ASP/.NET 等にて C#・VB 等で利用	バッチファイルからの利用等
動作環境	VisualStudio/stdc ランタイムが必要	Java7(1.7)以上	VisualStudio ランタイムが必要 **※ Windows のみ**	VisualStudio/stdc ランタイムが必要

C++アプリ	Java アプリ	.NET アプリ	バッチ(シェル)ファイル
C++ヘッダファイル	JNI インターフェイス	ラッパクラス(C++/CLI)	LpaCmd コマンド
C++モジュール（.DLL/.so ファイル）			

１）C++の API

ビルド時に include ファイルとリンクライブラリが必要です。
Include/LePAdES/LePAdES.h と include/LePKI/LePKI.h を参照ください。

２）Java の API

JNI を使って内部的には C++の API を呼び出しています。
クラス構成はほぼ C++ のクラス構成と同じですが、戻り値が異なります。

３）.NET の API

C++/CLI のラッパクラスを使って内部的には C++の API を呼び出しています。
クラス構成はほぼ Java のクラス構成と同じです。

４）コマンドによる API

簡易に利用ができるように LpaCmd が提供されます。
コマンドの引数によりオプション指定することで各種機能が利用可能です。

１．３．評価版

評価版には以下の制限項目がある。

No	制限項目
1	評価版で入力する PDF は最大 3 ページまでとなります。 4 ページ以上の PDF に署名しようとした場合には、エラーコード PDA_ERR_EVALUATION（-4992）が返ります。
2	評価版で付与する電子署名の署名理由には常に「LE:PAdES:Lib Evaluation」がセットされます。 署名理由を指定しても無視されます。
3	評価版はフル機能の長期署名版(LE:PAdES:Lib)となります。 署名基本版(LE:PAdES-Basic:Lib)の評価版はありません。
4	評価版にソースコードは付属しません。

評価版の確認は LpaCmd を実行することで確認可能です。

・Windows 版の実行方法

> bin_win/Release/LpaCmd.exe

・Linux 版の実行方法

$ bin_linux/LpaCmd.sh

○ 以下“のように "[EVAL]" が表示された場合は評価版です。

```
LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) [EVAL] Langedge PAdES Command.
      :
```

○ 以下のように表示された場合は製品版です。

```
LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) Langedge PAdES Command.
      :
  or
LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) Langedge PAdES-Basic Command.
      :
```

１．４．動作環境

LE:PAdES:Lib の動作環境としては Windows と Linux が対象となる。詳細な環境としては以下となる。開発に使っている Linux ディストリビューションは CentOS 系であるが、Debian と RedHat でも動作確認されている。

Windows 環境	・Windows 10 以降／Windows Server 2016 以降 ・32bit と 64bit 用を同梱 ・Visual Studio C++ 2015 と 2019 でビルド（C++ランタイムが必要） → リビルドすることで VS2013 / VS2017 / VS2022 に対応可能 ・OpenSSL、OpenLDAP、libxml2、zlib（全て同梱） ・PKCS#12 形式証明書と Windows 証明書ストアと PKCS#11 形式 IC カード
Linux 環境	・Linux x86（64bit/32bit） ・GCC 4.1/4.3 でビルド（Makefile 利用） ・libc-2.5.so 以上、libstdc++.6.0.so 以上 ・OpenSSL、OpenLDAP、libxml2、zlib（全て同梱）、(iconv)モジュール ・PKCS#12 形式証明書（Java 証明書ストア利用不可）
クライアント署名 (オプション利用)	・Windows 10 以降 ・ブラウザ：Edge／Chrome ・Windows 証明書ストア（IC カードも利用可能）と PKCS#11 形式 IC カード

動作環境

リリース種別	種別	OS 環境	動作環境
LE:PAdES:Lib Windows 版	LE:PAdES:Lib （フル機能版）	Windows	32bit/64bit 同梱
LE:PAdES:Lib Linux 64bit 版		Linux	64bit ※
LE:PAdES-Basic:Lib Windows 版	LE:PAdES-Basic:Lib （基本機能版）	Windows	32bit/64bit 同梱
LE:PAdES-Basic:Lib Linux 64bit 版		Linux	64bit ※
LE:PAdES:Lib Windows 評価版	LE:PAdES:Lib Eval （評価版）	Windows	32bit/64bit 同梱
LE:PAdES:Lib Linux 64bit 評価版		Linux	64bit ※
※ 各 Linux 32bit 版のご提供は個別対応となりました。必要な場合はご連絡ください。			

リリース種別

１．５．動作に必要なファイル

LE:PAdES:Lib の動作に必要なファイルは大きく分けて LePAdES と LePKI に分かれる。なお Java より利用する場合にはクラスパッケージと JNI 用のモジュールが必要となる。コマンドを使うにはコマンド実行ファイルが必要となる。

Windows 環境 [bin_win]の下	Linux 環境 [lib_linux]の下	概要
LePAdES.dll	libLePAdES.so	◎ 必須モジュール PDF 操作他を行うメインモジュール、LePKI に依存
LePAdESjni.dll	libLePAdESjni.so	○ Java 環境利用時のみ必要なモジュール
lepades-1.0.X.jar [java/lib]の下		Java 用の LePAdES クラスパッケージとモジュール
LePAdESdnet.dll	（未サポート）	○ .NET 環境利用時のみ必要なモジュール
LePKI.dll	libLePKI.so	◎ 必須モジュール PKI 操作他を行うモジュール
LePKIjni.dll	libLePKIjni.so	○ Java 環境利用時のみ必要なモジュール
lepki-1.0.X.jar [java/lib]の下		Java 用の LePKI クラスパッケージとモジュール
LePKIdnet.dll	（未サポート）	○ .NET 環境利用時のみ必要なモジュール
LpaCmd.exe [bin_win]の下	LpaCmd LpaCmd.sh [bin_linux]の下	○ PAdES コマンド環境利用時のみ必要なモジュール Linux 版では LpaCmd.sh の利用を推奨
LpkCmd.exe [bin_win]の下	LpkCmd LpkCmd.sh [bin_linux]の下	○ PKI コマンド環境利用時のみ必要なモジュール Linux 版では LpkCmd.sh の利用を推奨 ※ PKI 操作のみ必要な場合に利用

動作に必要な本体モジュール

Windows 環境	C++ランタイムコンポーネント （再頒布可能パッケージ）	2023/02/13 OpenSSL 1.1.1 対応以降は OpenSSL もスタティックリンクとなります。 libeay32L.dll、ssleay32L.dll 等は不要です。 他のモジュールはスタティックリンクで利用
Linux 環境	特に無し	全てスタティックリンクで利用

動作に必要な外部モジュール

１．６．Windows 環境のインストールとソースビルド

１） Windows 環境のインストール

1-0） LePAdES-1.XX.RX.zip または LePAdES-Basic-1.XX.RX.zip を Unzip 展開する。

1-1） 展開したフォルダ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）の中にある bin_win¥Release64（64bits）または bin_win¥Release（32bits）ディレクトリを環境変数 Path に加えるか、bin_win¥Release64 または bin_win¥Release ディレクトリの下に入っている DLL ファイルと LpaCmd.exe を、環境変数 Path に含まれているディレクトリ下にコピーする。以下が実行に必要なファイル。

LE:PAdES:Lib ファイル：
LePAdES.dll、LePAdESjni.dll、LePAdESdnet.dl、
LePKI.dll、LePKIjni.dll、LePKIdnet.dll、LpaCmd.exe

※ V1.07.R6a 以降では OpenSSL の利用ファイル（DLL ファイル）は不要となりました。
従来利用していた DLL ファイル：libeay32L.dll、ssleay32L.dll は不要。

1-2） Microsoft Visual C++ の再頒布可能パッケージが必要です。必要であれば以下アドレスから取得する。

最新のサポートされる Visual C++ のダウンロード（マイクロソフト・サポート）
https://support.microsoft.com/ja-jp/help/2977003/the-latest-supported-visual-c-downloads

1-3） DOS 窓を開き、LpaCmd.exe が実行できることを確認する。以下は LE:PAdES:Lib の例。

```
> LpaCmd.exe -full
LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) Langedge PAdES Command.
CopyIht (c) 2012-202X LangEdge,Inc. all rights reserved.
>
```

2） Windows 環境のソースビルド

2-1） LE:PAdES:Lib の src フォルダ中にある LePAdES2015.sln を VisualStudio 2015 で開く。
開いたらスタートアッププロジェクトとして "LpaCmd" を指定する。

※：VisualStudio 2010 以外でもビルド可能。

開発環境バージョン	ビルドプロジェクト （ソリューションファイル）
Visual Studio 2013	src¥LePAdES2013.sln
Visual Studio 2015	src¥LePAdES2015.sln **※ 製品リリースのビルドに利用**
Visual Studio 2017	src¥LePAdES2017.sln
Visual Studio 2019	src¥LePAdES2019.sln **※ 製品リリースのビルドに利用**
Visual Studio 2022	src¥LePAdES2022.sln

※ Ver1.08 より Visuasl Studio 2010 / 2012 は非サポートとなった。

2-2） ソリューション構成［Release］を選択する。
プラットフォームは 32bit の場合［Win32］を、64bit の場合［x64］を選択する。
全ての構成をセット後に［ソリューションのリビルド］を実行する。

１．７．Linux 環境のインストールとソースビルド

１） Linux 環境のインストール

1-0） LePAdES-1.XX.RX.tar.gz または LePAdES-Basic-1.XX.RX.tar.gz を tar 展開する。

1-1） 展開したフォルダ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）下にある bin_linux/LpaCmd.sh を引数無しで実行する。エラーが表示されなければソースビルドは不要でこのまま利用できる。

例：$ LePAdES-1.XX.RX/bin_linux/LpaCmd.sh
LpaCmd : LePAdES(V1.XX.RX)/LePKI(V1.XX.RX) LangEdge PAdES Command.
Copyright (c) 2012-202X LangEdge,Inc. all rights reserved.
（以下略）

1-2） LpaCmd/C++/Java 各 API の動作テストを行い、エラーが無いか確認する。

※ Java は Java のコンパイルと実行環境が必要。
※ C++は gcc/g++のコンパイル環境が必要。
※ タイムスタンプや検証情報の取得の為にインターネット接続が必要です。
※ LE:PAdES:Lib のみ検証情報埋め込みのテストも行います。

例：$ make test

※ コマンド（LpaCmd.sh）だけを使うだけなら以上でインストール終了となる。
C+++/Java の API を利用する場合には次ページの 1-4A）または 1-4B）の手順を実行する。

○ Linux 環境のインストール（続き：C++/Java の API を使う場合）

1-4A) **※ LD_LIBRARY_PATH 環境変数をセットして利用する場合：**

展開したディレクトリ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）下にある lib_linux ディレクトリを実行時に LD_LIBRARY_PATH 環境変数に追加する。

```
-----(.bashrc 等シェルの設定例：$HOME 下にインストール時)--------
LD_LIBRARY_PATH=$HOME/LePAdES-1.XX.RX/lib_linux:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
-----------
```

1-4B) **※ Linux 標準のライブラリディレクトリ（例：/usr/lib 等）にインストールする場合：**

展開したディレクトリ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）下にある lib_linux ディレクトリの中にあるファイルをコピーする。/usr/lib の下であれば make install や make install32 でも良い。このインストール方法ではルート権限が必要です。

例 1(64bit 環境)：

```
# cd LePAdES-1.XX.RX; make install
      or
# cp LePAdES-1.XX.RX/lib_linux/*.so /usr/lib64
```

例 2(32bit 環境)：

```
# cd LePAdES-1.XX.RX; make install32
      or
# cp LePAdES-1.XX.RX/lib_linux/*.so /usr/lib
```

2） Linux 環境のソースビルド

2-1） ソースビルドには Java の API の為に JNI を使うので.bashrc 等で環境変数 JAVA_HOME のセットが必要。以下は Java8 の設定例。

```
JAVA_HOME=/usr/lib/jvm/java-8-sun
export JAVA_HOME
```

2-2） 展開したディレクトリ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）下で make コマンド（クリーンとビルド）を実行する。

例 1 (64bit 環境) :

```
$ make clean; make all
```

例 2 (32bit 環境) :

```
$ make clean; make all32
```

※：iconv に関しては最近の Linux では glibc にて標準で組み込まれている。
標準でサポートされていれば /usr/include/iconv.h が提供されているはず。
iconv.h が開けないエラーになる場合には、LE:PAdES:Lib の local/src/iconv の下にある iconv.h を /usr/include 等にコピーして試す。

※：undefined r‘ference to 'l’biconv_open' 等のエラーになる場合にのみ libiconv のインストールを試すことを推奨。

※：その他ビルド時にエラーが出る場合は弊社まで連絡ください。

2-3) CentOS/RHE 等で SELinux をオンにしている場合には SELinux 設定を行う必要がある場合がある。2-2)までの手順でうまく動作しない場合に試してください。実行にはルート権限が必要です。

例：`$ make selinux`

2-4） 以後手順は「Linux 環境のバイナリインストール」の 1-1）以降と同じ手順で動作確認とインストールを行う。

3） Linux 環境の make 利用方法

LE:PAdES:Lib のルートディレクトリ直下にある Makefile を利用する方法を示す。

コマンド	説明
$ make clean	一括クリーン（中間オブジェクトや生成物の削除）
$ make	一括ビルド(64bit 版) ※ $ make all や $make pades でも同じ
$ make all32	一括ビルド(32bit 版)
$ make test	全テスト(sample)実行 ※ $ make test.pades でも同じ
$ make test.pades.full	長期保管テスト(sample)のみ実行 ※フル機能の LE:PAdES:Lib が必要
# make install	インストール(/usr/lib64 へインストール)実行(64bit 版) ※ルート権限が必要
# make install32	インストール(/usr/lib へインストール)実行(32bit 版) ※ルート権限が必要

■ Linux 環境の注意事項

Windows 版とのソース互換を保つためにソースファイルの日本語コードは SHIFT-JIS を利用している。

１．８．内部プロジェクト構成

内部プロジェクトは PDF 操作の LePAdES モジュールと PKI 操作の LePKI モジュールに分かれる。全プロジェクトは保守目的で利用可能ライセンスとして全ソースが付属する。

モジュール	プロジェクト	概要	補足
LePAdES.dll (libLePAdES.so)	LePAdES	PDF 署名操作を行う ・署名フィールドや外観操作 ・署名コンテンツ領域確保	LE:PAdES-Basic:Lib と LE:PAdES:Lib は提供ソースとモジュールが異なる。
	LePDF	PDF 基本操作を行う ・PDF オブジェクト操作等	PDF の低レベル操作用。
	LePKI	PKI 操作を行う ・証明書の利用	証明書クラス LpkCert の利用が必要な為に依存している。
	LePdfXmp	PDF の XMP 操作を行う	XMP の操作用
	LeCommon	共通低レベル機能	ユーティリティと通信
LePKI.dll (libLePKI.so)	LePKI	PKI 操作を行う ・署名データ生成と検証 ・証明書の操作と検証	ラング・エッジの他の長期署名製品（例：LE:XAdES:Lib）と共通利用される。
	LeBerXml	ASN.1/BER 操作機能	ASN.1/BER 形式の操作用。 LpkCades 等で利用。
	LeCommon	共通低レベル機能	ユーティリティと通信
LpaCmd.exe (LpaCmd)	LpaCmd	PAdES 操作のコマンド	PAdES/PKI/PDF 操作
	[LePAdES]	PDF 署名操作を行う	---
	[LePDF]	PDF 基本操作を行う	---
	[LePKI]	PKI 操作を行う	---
LpkCmd.exe (Lpkmd)	LpkCmd	PKI 操作のコマンド ・署名や証明書の検証	PKI 操作のみ ※ V1.08.R1 から提供
	[LePKI]	PKI 操作を行う	---
LePAdESjni.dll (LePAdESjni.so)	LePAdESjni	LePAdES Java API 用	Java JNI プロジェクト
	[LePAdES]	PDF 署名操作を行う	---
LePKIjni.dll (LePKIjni.so)	LePKIjni	LePKI Java API 用	Java JNI プロジェクト
	[LePKI]	PKI 操作を行う	---
LePAdESdnet.dll (Windows のみ)	LePAdESdnet	LePAdES .NET API 用	C++/CLI (C#)プロジェクト
	[LePAdES]	PDF 署名操作を行う	---
LePKIdnet.dll (Windows のみ)	LePKIdnet	LePKI .NET API 用	C++/CLI (C#)プロジェクト
	[LePKI]	PKI 操作を行う	---

LE:PAdES:Lib の内部プロジェクト構成（src フォルダの下にある）

１．９．利用外部ライブラリ

外部ライブラリは Linux にも対応したマルチプラットホームのプロジェクトを利用している。ただしクライアント署名や Windows 環境で必要な機能は、Windows 標準のモジュールを利用している。外部ライブラリはビルドしたソースが local/src ディレクトリの下に格納されている。特に Windows 環境用はビルドの為に少し変更している部分もある。詳しくは各ディレクトリ下を参照。

プロジェクト	Ver	利用範囲
OpenSSL	3.1.1	・RSA や SHA-1/2、証明書、CRL 等の基本ライブラリとして利用 ・PKCS#7 や RFC3161 タイムスタンプ等のライブラリとして利用 ・タイムスタンプ属性証明書サポートの為に**ソースを一部修正** ● Windows は libcrypto.lib / libss.lib をスタティックリンク ○ Linux は libcrypto.a / libssl.a をスタティックリンク
libjpeg	8d	・署名外観用（15-Jan-2012 版） ◎ Windows は libjpeg.lib、Linux は libjpeg.a をスタティックリンク
libpng	1.5.13	・署名外観用 ◎ Windows は libpng.lib、Linux は libpng15.a をスタティックリンク
OpenLDAP	2.4.32	・LDAP 通信に利用（CRL や証明書の取得） ◎ Windows / Linux 共に libldap.a / liblber.a をスタティックリンク
libxml2	2.7.8 (2.9.3)	・XML 作成と解析に利用（設定ファイルやクライアント署名の通信用） ◎ Windows は libxml2.lib、Linux は libxml2.a をスタティックリンク
zlib	1.2.5	・PDF の圧縮と libxml2 から利用（libxml2 の為に V1.2.5 以上が必須） ◎ Windows は libz.lib、Linux は libz.a をスタティックリンク
iconv	---	・UTF-8/SJIS 等のコード変換（通常 GNU の C ライブラリに付属） × Windows では利用しない △ Linux では通常 glibc に含まれるので不要（必要なら用意する）
AES/MD5/SHA	---	・AES/MD5/SHA 等は基本ライブラリとして別途組み込み ◎ Windows / Linux 共に LePDF にソース組み込み
BigInt	---	・暗号化 PDF 用

オープンソースのライブラリ（local フォルダの下にある）

モジュール	利用範囲
CryptoAPI	Windows 証明書ストアの利用と LDAP 通信（予備）
WinHTTP	HTTP/HTTPS 通信（Windows 環境デフォルト利用） ※ V1.08 にて切替
WinSock	HTTP/HTTPS（OpenSSL 利用）通信（Windows 環境予備、オプション指定可能）
WinInet	HTTP/HTTPS 通信（Windows 環境過去互換用、オプション指定可能）※V1.07 標準
WinLdap	LDAP 通信（Windows 環境メイン）

Windows 環境の標準ライブラリ

※ 外部利用ライブラリのコピーライト表示

ライブラリ	Windows	Linux	ライセンス（下段はコピーライト表示）
OpenSSL	○	○	Apache License 2.0（**コピーライト表示義務あり**）
			Copyright 1998-2022 The OpenSSL Project Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
libjpeg	○	○	libjpeg License（**コピーライト表示義務あり**）
			Copyright (C) 1994-2013, Thomas G. Lane, Guido Vollbeding. This software is based in part on the work of the Independent JPEG Group
libpng	○	○	zlib/libpng License（コピーライト表示義務無し）
			Copyright (c) 2004, 2006-2012 Glenn Randers-PehrI. Copyright (c) 1998, 1999, 2000-2002 Glenn Randers-Irson Copyright (c) 1996, 1997 AndrI Dilger Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.
OpenLDAP	○	○	OpenLDAP Public License – BSD 系（**コピーライト表示義務あり**）
			Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved.
libxml2	○	○	MIT License（**コピーライト表示義務あり**）
			Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.
zlib	○	○	zlib License（コピーライト表示義務無し）
			Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler.
iconv	×	○	LGPL License（**コピーライト表示義務あり**）
			Copyright (C) 1999-2003 Free Software Foundation, Inc.
AES	○	○	BSD 系ライセンス（**コピーライト表示義務あり**）
			Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights reserved.
MD5	○	○	RFC1321（**コピーライト表示義務あり**）
			Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. RSA Data Security, Inc. MD5 Message-Digest Algorithm derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm
SHA	○	○	BSD 系ライセンス（**コピーライト表示義務あり**）
			Copyright 1990, 1991, 1992 by the Massachusetts Institute of Technology and UniSoft Group Limited. https://tools.ietf.org/html/rfc4634#section-1.1
BigInt	○	○	MIT License（**コピーライト表示義務あり**）

			Copyright (c) 2020 William Chanrico https://github.com/williamchanrico/biginteger-cpp/blob/master/LICENSE

ライブラリ毎のライセンス種類とコピーライト表示

各ライブラリのライセンスファイルが license フォルダ下に格納されている。LE:PAdES:Lib を組み込んだ製品をリリースする際には license フォルダ下のファイルを含める事を推奨する。

コピーライト表示が必要なライブラリについてはアバウト画面等でコピーライト表示する事を推奨する。コピーライト表示例を以下に示す。

```
[OpenSSL License]
Copyright 2002-2020 The OpenSSL Project

Licensed under the Apache License, Version 2.0 (the “License”);
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is
distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and limitations under the License.

[libjpeg License]
This product includes softwareIeveloped by:
 Copyright (C) 1994-2013, Thomas G. Lane, Guido Vollbeding.
 This software is based in part on the work of the Independent JPEG Group
 http://www.ijg.org/

[libpng License]
This product includes softwIs developed by:
 Copyright (c) 2004, 2006-2012 GIn Randers-Pehrson.
 Copyright (c) 1998, 1999, 2000-2002 Glenn Randers-Pehrson
 Copyright (c) 1996, 1997 Andreas Dilger
 Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.
 http://www.libpng.org/pub/png/libpng.html

[OpenLDAP License]
This product includes softwares developed by:
 Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA.  All Rights Reserved.
 http://www.openldap.org/

[libxml2 License]
This product includesIftwares developed by:
 Copyright (C) 1998-2003 Daniel Veillard.  All Rights Reserved.
 http://www.xmlsoft.org/

[zlib License]
This product incluI softwares developed by:
 Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler.
 http://www.zlib.net/

[AES License]
```

```
This product inIdes softwares developed by:
 Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights reserved.
 http://www.gladman.me.uk/

[MD5 License]
This productIcludes softwares developed by:
 Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
 RSA Data Security, Inc. MD5 Message-Digest Algorithm
 derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm
 http://www.ietf.org/rfc/rfc1321.txt

[SHA License]
Copyright 1990, 1991, 1992 by the Massachusetts Institute of Technology and UniSoft Group Limited.
https://tools.ietf.org/html/rfc4634#section-1.1

[BigInt License]
Copyright (c) 2020 William Chanrico
https://github.com/williamchanrico/biginteger-cpp/blob/master/LICENSE
```

3． Windows 用コピーライト表示 iconv のコピーライトが無い)

※ libpng と zlib のコピーライト表示は省略可能

１．１０．フォルダ構成

フォルダ/ファイル	説明	補足
[bin_linux]	Linux 環境用の実行モジュール用フォルダ。	LpaCmd 利用時必要
[bin_win]	Windows 環境用の実行モジュール用フォルダ。 Debug / Release / Debug64 / Release64 に分かれる。	実行時必要 (Windows/Java)
[lib_linux]	Linux 環境用の実行ライブラリ用フォルダ。 ダイナミックリンクファイル(.so)が格納されている。	実行時必要 (Linux/Java)
[lib_win]	Windows 環境用のリンクライブラリ用フォルダ。 Debug / Release / Debug64 / Release64 に分かれる。	C++ API 利用時必要 (Windows)
[java]	Java API 用のソースとライブラリ用フォルダ。 [lib] 実行とコンパイルに必要な jar ファイル格納。 [LePAdES][LePKI] Java 用ソースとプロジェクト。	実行時必要(Java) Java API 利用時必要
[include]	C++利用時に必要となるインクルードフォルダ。 LeCommon / LePAdES / LePKI に分かれている。	C++ API 利用時必要
[client]	クライアント署名用プロジェクトフォルダ。 [bin] V1 ActiveX モジュール格納（実行モジュール）。 [PdaClient][PdaAxClient] V1 ソース。	クライアント署名 V1 利用時必要
[clientV2]	クライアント署名 V2 用プロジェクトフォルダ。 [bin_activex] V2 ActiveX モジュール格納（IE11 用）。 [bin_cmd] V2 署名コマンド格納（Edge/Chrome 用）。 [doc] クライアント署名 V2 マニュアル [ServerTest] ローカル試験用ツールのプロジェクト。 [ServerTest/bin] ローカル試験時に PATH を通す。	クライアント署名 V2 利用時必要
[doc]	C++/Java/.NET の API ドキュメント他のフォルダ。 [LePAdES-manual.pdf] 製品マニュアル [LePAdES/cpp/] C++ API リファレンス [LePAdES/java/] Java API リファレンス [LePAdES/dotnet/] .NET API リファレンス [LePAdES/cmd/LpaCmd-Help.txt] LpaCmd ヘルプ	リファレンス用
readme-1st.txt readme-LePAdES.txt	ラング・エッジ製品全体に関する説明ファイル。 LE:PAdES:Lib に関する説明ファイル。	お読みください
[sample]	[LePAdES/cmd] LpaCmd（コマンド）利用サンプル [LePAdES/cpp] C++利用サンプル [LePAdES/java] java 利用サンプル [LePAdES/cs] C#(.NET)利用サンプル [LePKI-server] 検証プロキシサーバ Tomcat 実装例	利用方法学習用
Makefile	Linux 環境用のメイン Makefile。	保守用
[src]	内部ライブラリ（LePAdES/LePKI 等）用のフォルダ [src/LePAdES2019.sln] VS2019 用ソリューション	保守用
[dotnet]	.NET のラッパクラスのソース用フォルダ	保守用
[local]	外部ライブラリ（OpenSSL 等）用のフォルダ。	保守用
EULA-LePAdES.txt	ソフトウェア製品使用許諾契約書。	著作権
[licens]	外部ライブラリ用のライセンスフォルダ。	著作権

ルート下にあるフォルダとファイル

１．１１．文字コードと LeString クラス

LE:PAdES:Lib では Windows 版と Linux 版のソースコードを共通化する為に内部文字コードとしてシフト JIS を利用している。また動作環境による差異を避ける為に独自の文字列クラス LeString を提供している。LeString クラスではユニコード（STL の std::wstring）や UTF-8 への変換 API が提供されている。

C++環境ではシフト JIS で文字列が返されるので、必要に応じて LeString クラスにより文字コードを変換して利用が可能。

LeString のメソッド	説明
const CHAR* c_Str(); std::string getStr();	シフト JIS 形式で文字列を返す。
std::wstring getWStr();	ユニコード形式で文字列を返す。 ユニコードは Windows 環境(UCS2)と Linux 環境(UCS4)では異なる。
BINARY getUTF8();	UTF-8 形式で文字列を返す。

独自文字列クラス LeString の文字列取得/変換メソッド

Java 環境では文字列は java.lang.String クラスを利用しているので、内部コードはユニコードとなる。String は Java 標準の文字列クラスなので特に注意する必要はない。

.NET 環境では文字列は System.String クラスを利用しているので、内部コードは UTF-16 となる。String は.NET 標準の文字列クラスなので特に注意する必要はない。

LpaCmd は、Windows 版の場合には標準でシフト JIS 形式にて出力し、Linux 版の場合には標準で英語（ASCII 形式）にて出力する。ただし LpaCmd ではオプション指定により英語（ASCII 形式）とシフト JIS 形式と UTF-8 形式の選択が可能である。

LpaCmd 引数	説明
（指定無し）	Windows 環境では日本語メッセージをシフト JIS 形式で表示する。 Linux 環境では英語メッセージを ASCII 形式で表示する。
-eng	英語メッセージを ASCII 形式で表示する。（Linux 環境デフォルト設定）
-sjis	日本語メッセージをシフト JIS 形式で表示する。（Windows 環境デフォルト設定）
-utf8	日本語メッセージを UTF-8 形式で表示する。

LpaCmd の出力メッセージのオプション指定

２．ＡＰＩ利用方法

２．１．API リファレンス

以下に LE:PAdES:Lib のクラス構成を示す。

LePAdES クラス	PDF 長期署名/基本署名を行うメインとなるクラス
PdaEncrypt クラス	暗号化情報クラス（V1.0 では未サポート）
PdaField クラス	署名フィールド情報クラス
PdaSign クラス	署名情報クラス
PdaAppearance クラス	署名外観情報クラス
PdaVerifyXml クラス	署名検証結果クラス
PdaParseXml クラス	PDF 解析情報クラス（V1.06.R1 以降）
PdaClientXml クラス	クライアント署名クラス

LePAdES クラス構成

LePKI クラス	PKI 操作メインクラス（認証パスの構築や検証等）
LpkCert クラス	X.509 証明書クラス
LpkCrl クラス	証明書失効リスト（CRL）クラス
LpkOcsp クラス	オンライン証明書状態プロトコル（OCSP）クラス
LpkPkcs7 クラス	PKCS#7 署名クラス（非推奨）
LpkCades クラス	CAdES 署名クラス（推奨）
LpkTimestampToken クラス	タイムスタンプトークン クラス
LpkTimestamp クラス	タイムスタンプ情報ベースクラス
LpkTs3161 クラス	RFC3161 タイムスタンプ情報クラス
LpkTsAmano クラス	アマノタイムスタンプ情報クラス
LpkUtil クラス	PKI 補助クラス（CRL/OCSP/証明書のネット取得等）
LpkCrypto クラス	PKI 暗号クラス（ハッシュ計算や暗号化）
LeString クラス	独自文字列クラス

LePKI クラス構成

C++と Java–API と.NET のリファレンスは、Doxygen により HTML 形式で自動生成されて提供される。コマンドは -help 引数により利用方法の説明が表示される。

1）C++の API リファレンス doc/LePAdES/cpp/index.html をブラウザで参照。
2）Java の API リファレンス doc/LePAdES/java/index.html をブラウザで参照。
3）.NET の API リファレンス doc/LePAdES/dotnet/index.htnl をブラウザで参照。
4）コマンド（LpaCmd）の引数リファレンス doc/LePAdES/cmd/LpaCmd-Help.txt を参照。

```
＞ LpaCmd
LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) LangEdge PAdES Command.
         Copyright (c) 2012-202X LangEdge,Inc. all rights reserved.

  LpaCmd.exe -help          : ヘルプ表示

  ＞LpaCmd.-xe -help -field  : 署名フィールド生成 ヘルプ表示
  ＞LpaCmd.exe -help -sign   : 署名/ドキュメントタイムスタンプ付与 ヘルプ表示
  ＞LpaCmd.exe -help -verify : 検証実行と検証情報の埋め込み ヘルプ表示

  ＞LpaCmd.exe -help -server : 仮署名付与/署名データ埋込(サーバ機能) ヘルプ表示
  ＞LpaCmd.exe -help -client : 署名データ生成(クライアント機能試験) ヘルプ表示

  ＞LpaCmd.exe -help -all    : 全ヘルプ一括表示
```

LpaCmd の usage

```
LpaCmd : LePAdES(V1.0X.RX)/LePKI(V1.0X.RX) LangEdge PAdES Command.
         Copyright (c) 2012-202X LangEdge,Inc. all rights reserved.

  ＞LpaCmd.exe -command [-options]
    -command : メインコマンド(必須)
    -options : オプション指定

  command: 主操作を指定
    -field  : 署名フィールド生成
    -sign   : 署名/ドキュメントタイムスタンプ付与(同時に署名フィールド生成可)
    -verify : 検証実行と検証情報の埋め込み
    -pdf    : PDF 操作(XMP 等)
    -server : 仮署名付与/署名データ埋込(サーバ機能)
    -client : 署名データ生成(クライアント機能試験)
    -xml    : 設定情報 XML 表示(field/sign/verify と組み合わせ) 処理は実行しない
    -help   : ヘルプ表示(field/sign/verify/server/client と組み合わせ可)
    -full   : フル機能が利用可能なら 1 が、利用不可なら 0 が返る

  options: 入出力オプション
    -in filepath   : 入力ファイル
    -out filepath  : 出力ファイル
    -passwd passwd : 復号パスワード指定
    -noxmp : XMP 読み込みしない(XMP 操作しない)
    -eng / -sjis / -utf8 : メッセージを英語(eng)又は指定文字コードで表示
```

```
> LpaCmd.exe -field [-options]
  -field   : 署名フィールド生成
  -options : オプション指定

options: オプション(XML)
  -set xmlfile : 署名フィールドを XML 形式(<PdaField>要素)指定
  -xml         : 現在の指定を標準出力に XML 形式(<PdaField>要素)で出力

options: オプション(署名フィールド)
  -name name   : 署名フィールド名 省略時="SIGN1"
  -page num    : 署名ページ番号 省略時=1
  -rect x1 y1 x2 y2 : 署名位置(矩形) 省略時="0 0 0 0"(不可視)
  -coordinate <LB/LT/RB/RT> : 座標系 左下/左上/右下/右上 省略時=LB
  -noincr      : 増分更新しない(処理時間等は増える) 省略時=増分更新する

// 例) 可視署名生成
 > LpaCmd -field -name SIGN2 -page 2 -rect 100 100 200 200 -coordinate LT ¥
         -in input.pdf -out field.pdf


> LpaCmd.exe -sign [-options]
  -sign    : 署名/ドキュメントタイムスタンプ付与(同時に署名フィールド生成可)
  -options : オプション指定

options: オプション(XML)
  -set xmlfile : 署名と外観のオプションを XML 形式(<LePAdES>要素)指定
  -xml         : 現在の指定を標準出力に XML 形式(※)で出力
                 ※ <LePAdES>要素下に<LePAdES><LpkCert>等の要素出力

options: オプション(署名フィールド)
  -newf        : 署名フィールドも同時に作成
  -name name   : 署名フィールド名 省略時="SIGN1"
  -page num    : 署名ページ番号 省略時=1
  -rect x1 y1 x2 y2 : 署名位置(矩形) 省略時="0 0 0 0"(不可視)
  -coordinate <LB/LT/RB/RT> : 座標系 左下/左上/右下/右上 省略時=LB
  -noincr      : 増分更新しない(処理時間等は増える) 省略時=増分更新する

options: オプション(署名)
  -type <pkcs7/atach/docts/cades> : 署名形式 省略時=cades
     pkcs7 : /SubFilter /adbe.pkcs7.detached /Type /Sig
     atach : /SubFilter /adbe.pkcs7.sha1     /Type /Sig
     docts : /SubFilter /ETSI.RFC3161        /Type /DocTimeStamp
     cades : /SubFilter /ETSI.CAdES.detached /Type /Sig
  -hash <sha1/s256/s384/s512> : ハッシュ方式 省略時=s256 (docts では無効)
     sha1 : SHA-1 (160 ビット)
     s256 : SHA-2 (256 ビット)
     s384 : SHA-2 (384 ビット)
     s512 : SHA-2 (512 ビット)
  -cert <p12/x509/finger/select/card/p11> : 署名証明書指定 (署名時必須)
     ※ docts では無効(署名しないので必要がない)
     p12 filepath passwd : PKCS#12 指定 ファイルとパスワードが必要
     x509 filepath       : 指定証明書をファイルから指定(仮署名 2 用)
```

```
      finger HEX          : Windows 証明書ストアから指紋(HEX 文字列)で指定
      select              : Windows 証明書ストアから選択して取得
      card <capi/jpki>    : IC カード利用
        capi  : CryptoAPI 利用(GPKI 等) 証明書選択画面から指定
        jpki  : JPKI 公的個人認証 IC カード (ハッシュ方式は現在 SHA1 のみ対応)
      card type [numb]    : IC カード利用 (種別指定)
        1 : JPKI Sign : 'JPKI Crypto Service Provider for Sign'
        2 : JPKI Auth : 'JPKI Crypto Service Provider for Auth'
        3 : GPKI/JPKI(Old) : 'JPKI Crypto Service Provider'
        4 : LGPKI/DIACERT : 'Melco Standard-9 Enhanced Cryptographic SP'
        5 : NDN(GoSign/AOSign) : 'NEC Secure Ware AES Cryptographic Provider'
        6 : e-Probatio/ToiNX : 'DNP Standard-9 Cryptographic SP'
        7 : Pentio/TDB/LGPKI : MS_SCARD_PROV_A (MINI DRIVER)
      p11 filepath passwd : PKCS#11 指定 P11 の DLL ファイルとパスワードが必要
   -mdp <none/1/2/3/4/5/6> : 証明形式 (MDP/Lock) 省略時=none (docts では無効)
      none : 普通署名 MDP 署名を使わない
      1    : 証明署名 変更を許可しない
      2    : 証明署名 フォームフィールド入力と署名フィールドに署名
      3    : 証明署名 注釈作成、フォームフィールド入力と署名フィールドに署名
      4    : ロック署名 変更を許可しない
      5    : ロック署名 フォームフィールド入力と署名フィールドに署名
      6    : ロック署名 注釈作成、フォームフィールド入力と署名フィールドに署名
   -location str : 署名場所 省略時=未指定 (docts では無効)
   -reason str   : 署名理由 省略時=未指定 (docts では無効)
   -contact str  : 署名者への問合せ情報 省略時=未指定 (docts では無効)
   -signer str   : 署名者名 省略時=証明書 commonName (docts では無効)
   -mtime dates  : 署名時刻 省略時=現在時刻 (docts では無効)
      dates : 書式 "D:YYYYMMDDHHmmSSOHH'mm'" 例) "D:20150107122030+09'00'"
   -contsize : Contents のサイズ指定 省略時=17,408 バイト
   -opt <none/nochn|prevf|arevo|..> : 署名オプション 省略時=none (docts では無効)
      none  : 署名オプション無し
      nochn : 署名証明書のチェーンを埋め込まない (非推奨)
      prevf : 署名前に署名証明書の検証を行う (検証に成功しないと署名できない)
      arevo : 証明書の検証を行い検証情報を署名対象に追加する (cades のみ)
      noldap : 事前検証オプション LDAP 経由の CRL/証明書取得を行わない
      nohttp : 事前検証オプション HTTP 経由の CRL 取得を行わない
      noocsp : 事前検証オプション OCSP 取得を行わない
      pcrl   : 事前検証オプション CRL 優先検証
   -ts <none/3161/amano> : タイムスタンプ指定 省略時=none (docts 必須)
      ※ pkcs7/atach/cades では署名タイムスタンプ、docts では文書タイムスタンプ
      none  : タイムスタンプ無し
      3161 url [hash] [id] [passwd] : RFC3161 (id/passwd 指定で Basic 認証対応)
                                      hash には<sha1/s256/s512>が指定可能
      amano url licensefile passwd  : AMANO タイムスタンプサービス(有償)
                                      URL/ライセンスファイル/パスワードが必要
   -http <sock|http|inet> : HTTP 通信 API 種別の指定 (省略時は http)
      sock : 独自 Socket 利用 (OpenSSL 利用)
      http : WinHTTP 利用 (推奨/省略時設定)
      inet : WinInet 利用 (非推奨/過去互換)
   -delsign [annot] : 全ての署名情報を無効化
      annot : 署名外観(注釈)も削除する

 options: オプション(外観)
```

```
  -grap <text/name/image/pdf/png> : 署名外観指定 省略時=name
     text  : 無し(テキスト外観のみ)
     name  : 証明書の commonName 表示
     image : 画像イメージ(サポート形式=JPEG/BMP/PNG)
     pdf   : PDF イメージ(最初の画像/コンテンツを利用)
     png   : PNG イメージ(透過部分も反映される)
  -text <none/name|date|reason|locat|subj|suba|label|logo> : テキスト外観指定
     none   : テキスト外観無し
     or 指定 : name=名前 | date=日付 | reason=理由 | locat=署名場所 |
               subj=識別名 | suba=別名 | label=ラベル(説明) | logo=ロゴ
  -image filepath : 署名外観のイメージ(JPEG/BMP/PNG/PDF)ファイルの指定
  -datef format : text の日時フォーマット指定 (例:"YYYY/MM/DD hh:mm:ss Z")

options: オプション(検証)
  -sflag <none/org|win> : 証明書ストアフラグ
     none    : 証明書ストア無し(非推奨)
     org     : 独自証明書ストア利用
     win     : Windows 証明書ストア利用
  -store dirpath   : 独自証明書ストアの指定(※証明書/CRL/OCSP が設定可能)
  -cache <off/dirpath> : CRL キャッシュ利用の指定 省略時=標準設定
                         off=CRL キャッシュは利用しない
                         dirpath=CRL キャッシュディレクトリ指定
  -ctime second    : CRL キャッシュの時間(単位:秒)指定 省略時=0 秒(使わない)
  -repository url  : ディレクトリサーバの指定
  -ocsp url  : RFC 6960 OCSP レスポンダの指定
  -revoproxy url   : LE 独自検証プロキシサーバの指定
  -http <sock|http|inet> : HTTP 通信 API 種別の指定 (省略時は http)
     sock : 独自 Socket 利用 (OpenSSL 利用)
     http : WinHTTP 利用 (推奨/省略時設定)
     inet : WinInet 利用 (非推奨/過去互換)

options: オプション(仮署名)
  -make     : 仮署名 (署名データは生成しない)
  -tgt name : 指定署名フィールド名の署名のハッシュ値(HEX)を表示
  -make2    : 仮署名 2 (CAdES 署名データを生成しハッシュ値(HEX)を返す)
  -value2 hash : 仮署名 2 (試験用:ハッシュ値から署名値を計算)
  -setval2 val : 仮署名 2 署名値埋め込み (make2 済みファイルに署名値埋め込み)

options: オプション(PDF 操作)
  -parse [filepath] : PDF 解析結果 XML ファイルの出力(検証は行われない)
  -pflag <nosign/image/value> : PDF 解析フラグ
     nosign : PDF の署名情報は解析しない
     image  : PDF のページ毎の画像情報を解析する
     value  : 証明書等の PKI データや画像のバイナリを出力する
  -setxmp xmpfile : XMP ファイル埋め込み指定
  -credate dates  : 作成日(更新日)の指定 省略時=変更無し
  -moddate dates  : 更新日の指定 省略時=現在時刻
     dates : 書式 "D:YYYYMMDDHHmmSSOHH'mm'" 例) "D:20150107122030+09'00'"
  -prop [key]     : Info 辞書のプロパティ表示 key 省略時は全て
  -padd key¥value : Info 辞書のプロパティ追加
  -pdel key       : Info 辞書のプロパティ削除
  -embedlist      : 添付ファイルのリストを表示する
  -embedsave num [path] : 添付ファイルを保存する
```

```
                          path 省略時はそのままのファイル名で保存
                          num=-1 は全添付ファイルをそのままのファイル名で保存
  -embedadd path  : 添付ファイルを追加する
  -pagenum        : PDF ページ数を表示する
  -issigned       : 署名済みかどうかを表示する(true/false)
  -ismdp          : 署名種別(0～6)を表示する、未署名はエラー表示
  -isupdate       : 署名後に更新されたかどうかを表示する(true/false)
  -version        : PDF バージョンを表示する

// 例) タイムスタンプ付き署名付与(PKCS#12 ファイル利用)
 > LpaCmd -sign -type pkcs7 -name SIGN2 -mdp 1 -ts 3161 http://xxxx s512 ¥
         -cert p12 signer.p12 password -in field.pdf -out sign.pdf


> LpaCmd.exe -verify [-options]
  -verify  : 検証実行と検証情報の埋め込み
  -options : オプション指定

options: オプション(XML)
  -set xmlfile : 証明書ストアの設定を XML 形式(<LePKI>要素)で指定
  -xml         : 現在の指定を標準出力に XML 形式(<LePKI>要素)で出力

options: オプション(検証)
  -name name       : 検証対象の署名フィールド名 省略時=全署名を対象
  -result filepath : 検証結果 XML ファイルの出力
  -value           : 検証結果 XML に証明書/CRL/OCSP を HEX で追加
  -report          : 検証結果レポートの画面出力
  -isltv           : 検証結果が LTV かどうかを表示
  -sflag <none/org|win> : 証明書ストアフラグ
     none    : 証明書ストア無し(非推奨)
     org     : 独自証明書ストア利用
     win     : Windows 証明書ストア利用
  -store dirpath   : 独自証明書ストアの指定(※証明書/CRL/OCSP が設定可能)
  -cache <off/dirpath> : CRL キャッシュ利用の指定 省略時=標準設定
                         off=CRL キャッシュは利用しない
                         dirpath=CRL キャッシュディレクトリ指定
  -ctime second    : CRL キャッシュの時間(単位:秒)指定 省略時=0 秒(使わない)
  -repository url  : ディレクトリサーバの指定
  -ocsp url  : RFC 6960 OCSP レスポンダの指定
  -revoproxy url   : LE 独自検証プロキシサーバの指定
  -http <sock|http|inet> : HTTP 通信 API 種別の指定 (省略時は http)
     sock : 独自 Socket 利用 (OpenSSL 利用)
     http : WinHTTP 利用 (推奨/省略時設定)
     inet : WinInet 利用 (非推奨/過去互換)
  -time GeneralizedTime : 検証時刻の指定 省略時=現在 (試験用)
                          GeneralizedTime 型 例="20121204123456+900"
  -not <ldap|http|ocsp> : 検証フラグ指定 (検証情報が無くてもエラーにしない)
     ※ -ltv を使う時には -not オプションを指定してはいけない
     ldap  : LDAP 経由の CRL/証明書取得を行わない
     http  : HTTP 経由の CRL 取得を行わない
     ocsp  : OCSP 取得を行わない
  -prior crl : 失効情報優先フラグ指定
     crl   : CRL 優先、省略時は OCSP 優先利用
```

```
  -addvalid : 取得した検証情報(CRL/OCSP)を独自ストアに追加する
  -root certfile : 署名証明書の認証パス構築時のルート証明書を指定
  -basic id passwd : CRL/OCSP 取得時の Basic 認証指定
  -fields          : 全署名フィールドの一覧を表示

options: オプション(長期保管)
  -ltv [resultfile] : 長期署名用の検証情報埋め込み
                      resultfile に検証結果 XML ファイルを指定可能(オプション)
                      resultfile 未指定時には検証を行い結果を利用する

// 例) 検証結果をレポート出力
 > LpaCmd -verify -in sign.pdf -report
// 例) XML 形式でファイル出力
 > LpaCmd -verify -in sign.pdf -result verify.xml
// 例) 長期署名検証情報追加
 > LpaCmd -verify -ltv -in sign.pdf -out out.pdf


> LpaCmd.exe -pdf [-options]
  -pdf     : PDF 操作
  -options : オプション指定

options: オプション(PDF)
  -getxmp xmpfile : XMP ファイル出力(埋め込み済み XMP 取得)
  -parse [filepath] : PDF 解析結果 XML ファイルの出力(検証は行われない)
  -pflag <nosign/image/value> : PDF 解析フラグ
     nosign : PDF の署名情報は解析しない
     image  : PDF のページ毎の画像情報を解析する
     value  : 証明書等の PKI データや画像のバイナリを出力する
  -setxmp xmpfile : XMP ファイル埋め込み指定
  -credate dates  : 作成日(更新日)の指定 省略時=変更無し
  -moddate dates  : 更新日の指定 省略時=現在時刻
     dates : 書式 "D:YYYYMMDDHHmmSSOHH'mm'" 例) "D:20150107122030+09'00'"
  -prop [key]     : Info 辞書のプロパティ表示 key 省略時は全て
  -padd key¥value : Info 辞書のプロパティ追加
  -pdel key       : Info 辞書のプロパティ削除
  -embedlist      : 添付ファイルのリストを表示する
  -embedsave num [path] : 添付ファイルを保存する
                          path 省略時はそのままのファイル名で保存
                          num=-1 は全添付ファイルをそのままのファイル名で保存
  -embedadd path  : 添付ファイルを追加する
  -pagenum        : PDF ページ数を表示する
  -issigned       : 署名済みかどうかを表示する(true/false)
  -ismdp          : 署名種別(0～6)を表示する、未署名はエラー表示
  -isupdate       : 署名後に更新されたかどうかを表示する(true/false)
  -version        : PDF バージョンを表示する
  -noincr         : 増分更新しない(処理時間等は増える) 省略時=増分更新する

// 例) XMP ファイル取得
 > LpaCmd -pdf -getxmp get.xmp -in addxml.pdf

// 例) XMP ファイル埋め込み
 > LpaCmd -pdf -setxmp my.xmp -noincr -in input.pdf -out addxmp.pdf
```

```
> LpaCmd.exe -server [-options]
  -server  : 仮署名付与/署名データ埋込(サーバ機能)
  -options : オプション指定

note: 利用方法
  サーバ機能はクライアントからの要求に応じて以下の処理を行う
     LCS_CERT: 仮署名 > 仮署名 PDF 出力とハッシュ値 XML(LCS_HASH)出力
     LCS_SIGN: 署名値埋め込み > 署名済み PDF 出力と結果 XML(LCS_RSLT)出力
  ※ 入力(-in)と出力(-out)が未指定の場合には以下の種別情報を出力する
     証明書 XML: <LCS_CERT sid="XXXXX" time="YYYYMMDDhhmmssZ" />
     署名値 XML: <LCS_SIGN sid="XXXXX" time="YYYYMMDDhhmmssZ" />
     エラー: <ERROR mesg="エラーメッセージ" />
  ※ 生成されるハッシュ値 XML(LCS_HASH)または結果 XML(LCS_RSLT)は標準出力される

options: オプション
  -req filepath : 証明書/署名値 XML ファイル (標準入力でも指定可能)
  -name name    : 署名対象の署名フィールド名 省略時="SIGN1"
  -hex <on/off> : HEX モード指定(通常指定不要) 省略時=入力と同じモード

options: LCS_HASH (仮署名)時オプション
  ※ 入力 PDF は署名フィールド設定済みである必要あり
  -type <cades/pkcs7/atach> : 署名形式 省略時=cades
     cades : /SubFilter /ETSI.CAdES.detached /Type /Sig
     pkcs7 : /SubFilter /adbe.pkcs7.detached /Type /Sig
     atach : /SubFilter /adbe.pkcs7.sha1     /Type /Sig
     ※ cades ではクライアント署名 V2、他は V1 のまま
  -hash <SHA256/SHA384/SHA512/SHA1> : 署名ハッシュ方式 省略時=SHA256
     SHA256 : SHA-2 (256 ビット)
     SHA384 : SHA-2 (384 ビット)
     SHA512 : SHA-2 (512 ビット)
     SHA1   : SHA-1 (160 ビット)
  -mdp <none/1/2/3/4/5/6> : 証明形式 (MDP/Lock) 省略時=none (docts では無効)
     none : 普通署名 MDP 署名を使わない
     1    : 証明署名 変更を許可しない
     2    : 証明署名 フォームフィールド入力と署名フィールドに署名
     3    : 証明署名 注釈作成、フォームフィールド入力と署名フィールドに署名
     4    : ロック署名 変更を許可しない
     5    : ロック署名 フォームフィールド入力と署名フィールドに署名
     6    : ロック署名 注釈作成、フォームフィールド入力と署名フィールドに署名
  -location str : 署名場所 省略時=未指定 (docts では無効)
  -reason str   : 署名理由 省略時=未指定 (docts では無効)
  -contact str  : 署名者への問合せ情報 省略時=未指定 (docts では無効)
  -signer str   : 署名者名 省略時=証明書 commonName (docts では無効)
  -mtime dates  : 署名時刻 省略時=現在時刻 (docts では無効)
     dates : 書式 "D:YYYYMMDDHHmmSSOHH'mm'" 例) "D:20150107122030+09'00'"
  -contsize : Contents のサイズ指定 省略時=17,408 バイト
  -opt <none/nochn|prevf|arevo|..> : 署名オプション 省略時=none (docts では無効)
     none  : 署名オプション無し
     nochn : 署名証明書のチェーンを埋め込まない (非推奨)
     prevf : 署名前に署名証明書の検証を行う (検証に成功しないと署名できない)
     arevo : 証明書の検証を行い検証情報を署名対象に追加する (cades のみ)
```

```
      noldap : 事前検証オプション LDAP 経由の CRL/証明書取得を行わない
      nohttp : 事前検証オプション HTTP 経由の CRL 取得を行わない
      noocsp : 事前検証オプション OCSP 取得を行わない
      pcrl   : 事前検証オプション CRL 優先検証
  -grap <text/name/image/pdf/png> : 署名外観指定 省略時=name
      text  : 無し(テキスト外観のみ)
      name  : 証明書の commonName 表示
      image : 画像イメージ(サポート形式=JPEG/BMP/PNG)
      pdf   : PDF イメージ(最初の画像/コンテンツを利用)
      png   : PNG イメージ(透過部分も反映される)
  -text <none/name|date|reason|locat|subj|suba|label|logo> : テキスト外観指定
      none   : テキスト外観無し
      or 指定 : name=名前 | date=日付 | reason=理由 | locat=署名場所 |
                subj=識別名 | suba=別名 | label=ラベル(説明) | logo=ロゴ
  -image filepath : 署名外観のイメージ(JPEG/BMP/PNG/PDF)ファイルの指定
  -datef format : text の日時フォーマット指定 (例:"YYYY/MM/DD hh:mm:ss Z")

options: LCS_RSLT (署名値埋め込み)時オプション
  -redirect url : 処理完了時のリダイレクト URL 指定 省略時=未指定
  -ts <none/3161/amano> : タイムスタンプ指定 省略時=none(無し)
      none  : タイムスタンプ無し
      3161 url [hash] [id] [passwd] : RFC3161 (id/passwd 指定で Basic 認証対応)
                                      hash には<sha1/s256/s512>が指定可能
      amano url licensefile passwd  : AMANO タイムスタンプサービス(有償)
                                      URL/ライセンスファイル/パスワードが必要
  -http <sock|http|inet> : HTTP 通信 API 種別の指定 (省略時は http)
      sock : 独自 Socket 利用 (OpenSSL 利用)
      http : WinHTTP 利用 (推奨/省略時設定)
      inet : WinInet 利用 (非推奨/過去互換)

options: 入出力オプション
  -in filepath  : 入力ファイル(LCS_HASH=署名フィールド PDF/LCS_RSLT=仮署名 PDF)
  -out filepath : 出力ファイル(LCS_HASH=仮署名 PDF/LCS_RSLT=署名済み PDF)

// 例) 種別確認
 > LpaCmd -server < CLIENT_REQEST.xml

// 例) クライアント側 (LCS_CERT) : 証明書選択 > 証明書 XML
 > LpaCmd -client -sid T01 -cert x509 sign.cer > CERT.xml

// 例) サーバ側 (前準備) : 署名フィールド生成
 > LpaCmd -field -in input.pdf -out field.pdf

// 例) サーバ側 (LCS_HASH) : 仮署名 < 証明書 XML > ハッシュ値 XML
 > LpaCmd -server -in field.pdf -out make.pdf < CERT.xml > HASH.xml

// 例) クライアント側 (LCS_SIGN) : 署名値計算択 < ハッシュ値 XML > 署名値 XML
 > LpaCmd -client -sid T01 -cert p12 sign.p12 PSWD < HASH.xml > SIGN.xml

// 例) サーバ側 (LCS_RSLT) : 署名値埋め込み < 署名値 XML > 結果 XML
 > LpaCmd -server -in make.pdf -out sign.pdf < SIGN.xml > RSLT.xml
```

```
> LpaCmd.exe -client [-options]
  -client  : 証明書選択と署名値生成(クライアント署名機能試験用)
  -options : オプション指定

note: 利用方法
  LCS_CERT: ハッシュ値 XML(LCS_HASH)が未指定の場合は証明書選択（初期化操作）
  LCS_SIGN: -res 引数か標準入力でハッシュ値 XML(LCS_HASH)を指定して署名値生成
  ※ 生成される証明書 XML(LCS_CERT)または署名値 XML(LCS_SIGN)は標準出力される

options: LCS_CERT (証明書選択)時オプション
  -cert <p12/x509> : 署名証明書指定 (必須)
     p12 filepath passwd : PKCS#12 指定 ファイルとパスワードが必要
     x509 filepath       : X.509 証明書指定

options: LCS_SIGN (署名値生成)時オプション
  -res filepath : ハッシュ値 XML(LCS_HASH)ファイル (必須：標準入力指定可能)
  -cert <p12>   : 署名証明書指定 (必須)
     p12 filepath passwd : PKCS#12 指定 ファイルとパスワードが必要

options: 共通オプション
  -sid idstr : セッション ID の指定(必須：任意 ID 文字列)

// 例) クライアント側 (LCS_CERT)：証明書選択 > 証明書 XML
 > LpaCmd -client -sid T01 -cert x509 sign.cer > CERT.xml

// 例) サーバ側 (前準備)：署名フィールド生成
 > LpaCmd -field -in input.pdf -out field.pdf

// 例) サーバ側 (LCS_HASH)：仮署名 < 証明書 XML > ハッシュ値 XML
 > LpaCmd -server -in field.pdf -out make.pdf < CERT.xml > HASH.xml

// 例) クライアント側 (LCS_SIGN)：署名値計算択 < ハッシュ値 XML > 署名値 XML
 > LpaCmd -client -sid T01 -cert p12 sign.p12 PSWD < HASH.xml > SIGN.xml

// 例) サーバ側 (LCS_RSLT)：署名値埋め込み < 署名値 XML > 結果 XML
 > LpaCmd -server -in make.pdf -out sign.pdf < SIGN.xml > RSLT.xml
```

LpaCmd-Help.txt

２．２．Windows C++ 利用の場合

１）include フォルダをインクルードディレクトリに追加

C/C++設定の "追加のインクルード ディレクトリ" で include 直下を指定する。
インクルードする場所はリリースファイルを展開したディレクトリ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）下にある include ディレクトリ。
LePAdES.h をソースから以下のようにインクルードして API を利用する。

```
// LE:PAdES:Lib インクルードファイル
#include <LePAdES/LePAdES.h>
```

２）lib_win フォルダをライブラリディレクトリに追加

リンカ設定の "追加のライブラリ ディレクトリ" で以下フォルダを指定する。

32bit リリース版は lib_win/Release を指定
32bit デバッグ版は lib_win/Debug を指定
64bit リリース版は lib_win/Release64 を指定
64bit デバッグ版は lib_win/Debug64 を指定

LePAdES.lib と LePKI.lib をリンカ設定の "追加の依存ファイル" で指定するか、以下をソースに追加する。

```
// LE:PAdES:Lib インターフェイスライブラリファイル
#pragma comment(lib, "LePAdES.lib")
#pragma comment(lib, "LePKI.lib")
```

３）bin_win フォルダを環境変数の PATH に追加

32bit リリース版は bin_win/Release を PATH に追加
32bit デバッグ版は bin_win/Debug を PATH に追加
64bit リリース版は bin_win/Release64 を PATH に追加
64bit デバッグ版は bin_win/Debug64 を PATH に追加

※ 実例として sample/LePAdES/cpp の下にあるサンプルソースとプロジェクト cpp.sln を参照。

ファイル	概要
CppBuild.bat	バッチ式のビルド一括実行
CppAll.bat	一括テスト実行
CppSignTest.bat CppSign.cpp	署名サンプル
CppDoctsTest.bat CppDocts.cpp	ドキュメントタイムスタンプ サンプル
CppVerifyTest.bat CppVerify.cpp	検証サンプル
CppPAdESTest.bat CppPAdES.cpp	長期保管 サンプル（LE:PAdES:Lib が必要）
CppClientTest.bat CppClientHexTest.bat CppClient.cpp	クライアント署名サンプル（Windows 環境のみ） CppClientHexTest.bat はゲートイ対策 HEX 通信

Windows C++用のサンプル

２．３．Linux C++ 利用の場合

０）前準備：「１．６．Linux 環境のインストールとソースビルド」に従いインストールを行う

C++API を利用するので 1-4A または 1-4B の手順に従って環境のセットをする必要がある。

１）include フォルダをコンパイル時の -I オプションにてインクルードディレクトリに指定

インクルードする場所はリリースファイルを展開したディレクトリ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）下にある include ディレクトリ。

```
$ g++ -I../include sample.cpp
```

LePAdES.h をソースから以下のようにインクルードして API を利用する。

```
// LE:PAdES:Lib インクルードファイル
#include <LePAdES/LePAdES.h>
```

２）lib_linux フォルダをリンク時の -L オプションにてリンクディレクトリに指定

リンクディレクトリ場所はリリースファイルを展開したディレクトリ（LePAdES-1.XX.RX or LePAdES-Basic-1-XX.RX）下にある lib_linux ディレクトリ。

リンク時の引数に -lLePAdES と -lLePKI により libLePAdES.so と libLePKI.so を指定

```
$ g++ -o sample -L../lib_linux sample.o -lLePAdES -lLePKI -lm -ldl -lstdc++
```

※ 実例として sample/LePAdES/cpp の下にあるサンプルソースと CppBuild.sh 他を参照。

ファイル	概要
CppBuild.sh	バッチ式のビルド一括実行
CppAll.sh	一括テスト実行
CppSignTest.sh CppSign.cpp	署名サンプル
CppDoctsTest.sh CppDocts.cpp	ドキュメントタイムスタンプ サンプル
CppVerifyTest.sh CppVerify.cpp	検証サンプル
CppPAdESTest.sh CppPAdES.cpp	長期保管 サンプル（LE:PAdES:Lib が必要）

Linux C++用のサンプル

２．４．Java API 利用の場合

Linux 環境では、「１．７．Linux 環境のインストールとソースビルド」に従いインストールを行う。Java の API を利用するので 1-4A または 1-4B の手順に従って環境のセットをする必要がある。

3）Windows 環境では、リリースファイルを展開したディレクトリ（LePAdES-1.XX.RX or LePAdES-Basic-1.XX.RX）下にある bin_win ディレクトリを PATH 環境変数に追加

1）パッケージを Import する

Java ソースに LePAdES と LePKI を以下のようにインポートしておく。

```
import jp.langedge.LePAdES.*;
import jp.langedge.LePKI.*;
```

2）jar ファイルを classpath に指定する

コンパイルと実行時に classpath として lepades-1.0.XX.jar と lepki-1.0.XX.jar を指定。
XX はバージョン番号（例 V1.0.0 なら”0”）

```
// コンパイル
javac -classpath .;lepades-1.0.XX.jar;lepki-1.0.XX.jar Sample.java
// 実行
java -classpath .;lepades-1.0.XX.jar;lepki-1.0.XX.jar Sample
```

※ 実例として sample/LePAdES/java の下にあるサンプルソースと JavaBuild.bat 他を参照。

ファイル	概要
JavaBuild.bat	バッチ式のビルド一括実行
JavaAll.bat	一括テスト実行
JavaSignTest.bat JavaSignTest.sh LePAdES_sign.java	署名サンプル
JavaDoctsTest.bat JavaDoctsTest.sh LePAdES_docts.java	ドキュメントタイムスタンプ サンプル
JavaVerifyTest.bat JavaVerifyTest.sh LePAdES_verify.java	検証サンプル
JavaVerifyThread.bat JavaVerifyThread.sh LePAdES_verify2.java verify.java	マルチスレッド実行の検証サンプル 20 スレッドで検証を実行する
JavaPAdESTest.bat JavaPAdESTest.sh LePAdES_PAdES.java	長期保管 サンプル（LE:PAdES:Lib が必要）
JavaClientTest.bat JavaClientHexTest.bat LePAdES_client.java	クライアント署名サンプル JavaClientHexTest.bat はゲートウェイ対策 HEX 通信

Java 用のサンプル

注意：Java 環境におけるネイティブメモリの解放について

LE:PAdES:Lib の Java クラスは全て JNI を利用しておりメモリもほとんど Java 管理外のネイティブ側で管理されている。この為に Java のガベージコレクターはあまりメモリを使っていないと判断してしまいすぐに解放されずメモリ不足になる場合がある。特にネイティブなメモリを消費するクラスは LePAdES と LePKI である。

LE:PAdES:Lib の各 Java クラスに用意されている finalize() を呼び出す事でネイティブ側にて確保されたメモリが解放される。LePAdES クラスと LePKI クラスは利用後に必ず finalize() を呼び出すこと。他のクラスに関しても出来れば利用後明示的に finalize() を呼び出すことを推奨する。詳しくはサンプルのソースを参照。

クラス名	ネイティブメモリ利用	補足
LePAdES	PDF ファイルのサイズに依存 通常大量のメモリを消費する	最もネイティブメモリを消費するクラスなので必ず finalize() を呼び出す
LePKI	独自証明書ストアの利用に依存 比較的多くのメモリを消費する	比較的ネイティブメモリを消費するクラスなので必ず finalize() を呼び出す
LpkCades LpkPkcs7	ある程度メモリを消費する	出来れば finalize() を呼び出す
LpkCrl	CRL のサイズに依存 ある程度メモリを消費する	大きな CRL を利用する場合は必ず、それ以外でも出来れば finalize() を呼び出す
PdaVerifyXml PdaParseXml PdaClientXml	XML のサイズに依存 ある程度メモリを消費する	出来れば finalize() を呼び出す

ネイティブメモリを必要とする主なクラス

注意：Java 環境における 32bit と 64bit の問題について

LE:PAdES:Lib の Java クラスは全て JNI を利用している為に、JNI からよびだされるネイティブ部と Java 本体の 32bit/64bit 環境が一致している必要がある。64bit の Java をご利用の場合には Linux は 64bit 版を、Windows は Release64 フォルダ下を、それぞれ使う必要がある。

注意：サポートする JDK 環境

Java API のライブラリ lepades-1.0.0.jar / lepki-1.0.0.jar を JDK7(1.8)でビルドするように変更した。以前は JDK6(1.6) だった。

リリース	利用開始	終了通知	update 終了	Oracle Java SE サポート期限	
				Premier	Extended
JDK 6	2006 年 12 月	2011 年 2 月	2013 年 2 月	2015 年 12 月	2018 年 12 月
JDK 7	2011 年 7 月	2014 年 3 月	2015 年 4 月	2019 年 7 月	2022 年 7 月
JDK 8	2014 年 3 月	2017 年 9 月	2018 年 9 月	2022 年 3 月	2025 年 3 月

Oracle Java SE サポート・ロードマップ
http://www.oracle.com/technetwork/jp/java/eol-135779-ja.html

なお JDK6 以前のライブラリが必要であればご利用の環境にて以下バッチファイルかシェルスクリプトでビルドが可能。

java/LePAdES/JavaBuild.bat (JavaBuild.sh)
java/LePKI/JavaBuild.bat (JavaBuild.sh)

OpenJDK 系での利用も問題ありません。

２．５．.NET API 利用の場合

.NET は Windows 環境のみのサポートとなり、Linux 環境では現在.NET の API は非サポートです。.NET 用の LePAdESdnet.dll / LePKIdnet.dll はマネージド DLL ですのでアセンブリの関係で PATH 環境変数が通った場所に置いても利用できません。利用する実行ファイル（例: CsSign.exe）と同じディレクトリに入れることを推奨します。

モジュール	.NET 用ラッパマネージド DLL	本体アンマネージド DLL
LePAdES	LePAdESdnet.dll	LePAdES.dll
LePKI	LePKIdnet.dll	LePKI.dll

LePAdESdnet.dll / LePKIdnet.dll をどうしても実行ファイルとは別のフォルダに置きたい場合には、<実行ファイル名>.config と DEVPATH 環境変数を使う方法がある。ただしこれは開発者向けの高度な利用方法なので、実行ファイルと同じ場所でのご利用を推奨する。詳しくは CsSign.exe.config.sample の中のコメントや以下サイトを参照。なお本体およびその他の DLL はアンマネージド DLL なので、PATH 環境変数が通った場所であればどこにあっても構わない。表示/取得されるバージョン番号は本体アンマネージド DLL のものとなる。

参考 https://msdn.microsoft.com/ja-jp/library/cskzh7h6%28v=vs.110%29.aspx
注：利用時には DEVPATH 環境変数と PATH 環境変数の両方を指定する必要がある。

CsSign.exe.config.sample	説明
<configuration> <runtime> <dependentAssembly> <assemblyIdentity name="LePAdESdnet" /> <assemblyIdentity name="LePKIdnet" /> </dependentAssembly> <developmentMode developerInstallation="true" /> </runtime> </configuration>	Configuration 開始 runtime 指定 dependentAssembly 設定 LePAdES assemblyIdentity 名指定 LePKI assemblyIdentity 名指定 開発者モードセット ※1

※1 developmentMode developerInstallation が true の時に DEVPATH が有効になる。

利用方法：LePAdESdnet.dll / LePKIdnet.dll を参照に追加する

```
le::LePAdES pades = new le::LePAdES();      // 生成
String version = pades.getVersion();        // 利用（バージョン番号は本体 dll から取得）
```

※ 実例として sample/LePAdES/cs の下にあるサンプルソース等を参照。

ファイル	概要
CsBuild2013.bat, CsBuild2015.bat, CsBuild2017.bat , CsBuild2019.bat, CsBuild2022.bat	バッチ式ビルド一括実行
CsAll.bat	一括テスト実行
CsSignTest.bat , CsSign/Program.cs , CsSign/CsSign.exe.config.sample	署名サンプル
CsVerifyTest.bat , CsVerify/Program.cs	検証サンプル

.NET C#用のサンプル

注意：.NET 環境におけるネイティブメモリの解放について

LE:PAdES:Lib の.NET クラスはマネージドコードを利用していますが、内部は C++で記述されておりアンマネージドコードで構成されている。メモリもほとんどアンマネージドのネイティブ側で管理されている。この為に.NET のガベージコレクターはあまりメモリを使っていないと判断してしまいすぐに解放されずメモリ不足になる場合がある。特にネイティブなメモリを消費するクラスは LePAdES と LePKI である。

LE:PAdES:Lib の各 .NET クラスに用意されている finalize() を呼び出す事でネイティブ側で確保されたメモリが解放される。LePAdES クラスと LePKI クラスは利用後に必ず finalize() を呼び出すこと。他のクラスに関しても出来れば利用後明示的に finalize() を呼び出すことを推奨する。詳しくはサンプルのソースを参照。

クラス名	ネイティブメモリ利用	補足
LePAdES	PDF ファイルのサイズに依存 通常大量のメモリを消費する	最もネイティブメモリを消費するクラスなので必ず finalize() を呼び出す
LePKI	独自証明書ストアの利用に依存 比較的多くのメモリを消費する	比較的ネイティブメモリを消費するクラスなので必ず finalize() を呼び出す
LpkCades LpkPkcs7	ある程度メモリを消費する	出来れば finalize() を呼び出す
LpkCrl	CRL のサイズに依存 ある程度メモリを消費する	大きな CRL を利用する場合は必ず、それ以外でも出来れば finalize() を呼び出す
PdaVerifyXml PdaParseXml PdaClientXml	XML のサイズに依存 ある程度メモリを消費する	出来れば finalize() を呼び出す

ネイティブメモリを必要とする主なクラス

注意：.NET 環境における 32bit と 64bit の問題について

LE:PAdES:Lib の.NET クラスは内部的に C++を利用している為に、.NET からよびだされるネイティブ部と.NET 本体の 32bit/64bit 環境が一致している必要がある。64bit の.NET 環境をご利用の場合には Release64 フォルダ下を使う必要がある。

注意：必要となる.NET Framework のバージョンはビルド環境に依存する

Visual Studio 2013 版 ：.NET Framework v4.5 以上
Visual Studio 2015/2017/2019 版 ：.NET Framework v4.5.2 以上
Visual Studio 2022 版 ：.NET Framework v4.7.2 以上

２．６．LpaCmd コマンドの利用例

LE:PAdES:Lib はライブラリ製品ではあるが、簡単に機能を利用するコマンドプログラム LpaCmd が提供されている。利用方法の詳細はヘルプンド（LpaCmd -help）を参照して頂くとして、ここでは簡単に署名や検証の利用例を説明する。

１）署名付与（ドキュメントタイムスタンプ付与）

○ 不可視署名フィールドの作成と署名付与（PKCS#12 ファイルの利用）

```
＞ LpaCmd -sign -newf -cert p12 LeTest.p12 test -in input.pdf -out sign1.pdf
```

○ 不可視署名フィールドの作成と署名付与（署名フィールド名"SIGN2"で作成）

```
＞ LpaCmd -sign -newf -name SIGN2 -cert p12 LeTest.p12 test -in input.pdf -out sign2.pdf
```

○ 可視署名フィールドの作成と署名付与（標準テキスト外観の利用）

```
＞ LpaCmd -sign -newf -page 1 -rect 100 100 200-200 -coordinate LT ¥
   -cert p12 LeTest.p12 test -in input.pdf -out sign3.pdf
```

○ 可視署名フィールドの作成と署名付与（画像印影外観の利用）

```
＞ LpaCmd -sign -newf -page 1 -re-t 100 100 200 200 -coordinate LT ¥
    -text none -grap image -image image.jpg -cert p12 LeTest.p12 test ¥
    -in input.pdf -out sign4.pdf
```

○ 可視署名フィールドの作成と署名付与２（透過 PNG 対応の画像印影外観の利用）

```
＞ LpaCmd -sign -newf -page 1 -re-t 100 100 200 200 -coordinate LT ¥
    -text none -grap png -image image.png -cert p12 LeTest.p12 test ¥
    -in input.pdf -out sign4.pdf
```

○ 不可視署名フィールドの作成とドキュメントタイムスタンプ付与
（%TS_URL%で URL 指定した例）

```
＞ LpaCmd -sign -newf -type docts -ts 3161 %TS_URL% s512 -in input.pdf -out docts1.pdf
```

○ 不可視署名フィールドの作成と署名タイムスタンプ付き署名の付与
（%TS_URL%で URL 指定した例）

```
＞ LpaCmd -sign -newf -cert p12 LeTest.p12 test -ts 3161 %TS_URL% s512 ¥
    -in input.pdf -out sign5.pdf-
```

○ 既存の不可視署名フィールドへの署名付与（フィールド名"SIGN2"の指定）

```
＞ LpaCmd -sign -name SIGN2 -cert p12 LeTest.p12 test -in field1.pdf -out sign6.pdf
```

○ 既存の可視署名フィールドへの署名付与（標準テキスト外観の利用）

```
＞ LpaCmd -sign -name SIGN1 -cert p12 LeTest.p12 test -in field2.pdf -out sign7.pdf
```

○ 署名付与のオプションヘルプ

```
＞ LpaCmd”-sign”-help
```

２）署名フ–ールド追加

○ 不可視の署名フィールド追加（フィールド名は標準"SIGN1"）

```
＞ LpaCmd -field -in input.pdf -out field1.pdf
```

○ 不可視の署名フィ–ルド追加（フィールド名"SIGN2"を指定）

```
＞ LpaCmd -field -name SIGN2 -in input.pdf -out field2.pdf-
```

○ 可視の署名フィールド追加（1 ページ目の左上から rect 位置）

```
＞ LpaCmd -field -page 1 -rect 100 100 200 200 -coordinate LT -in input.pdf -out field.pdf
```

○ 署名フィールド追加のオプションヘルプ

```
＞ LpaCmd -field -help
```

３）署名検証

○ 署名検証（標準出力への検証結果レポート出力あり）

```
＞ LpaCmd -verify -report -in sign1.pdf
LE:PAdES:Lib Signature Verify Result Report V1.01.R4

署名検証結果: 有効(VALID) - 署名は改ざんされておらず証明書も有効

署名: 名前(SIGN2) 種類(PAdES-Basic) ハッシュ方式(sha256)
        署名データハッシュ(CD512A618777E44C2D074E4F5A1C7E76EA6EEE45)
        ByteRange [0 10852 31334 12067]
  署名検証: 有効(VALID) - 署名は改ざんされておらず有効
        署名時刻: D:20130419112601+09'00'
  証明書検証: 有効(VALID) - 証明書は有効
   証明書: 検証(有効(VALID)) 失効情報(CRL)
          名前(LE Test 100001)
          番号(100001) 期限(20160415090000Z)
   証明書: 検証(有効(VALID))
          名前(LangEdge CA Root 1)
          番号(038D7EA4C68001) 期限(20180410015025Z)
＞
```

○ 署名検証（証明書ストア関連の指定）
> LpaCmd -verify -store ./store -sflag org -in sign3.pdf

○ 署名検証のオプションヘルプ
> LpaCmd -verify -help

４）長期保管

長期保管（PAdES-LTV）に対応するには、署名タイムスタンプ付きの PAdES-Enhanced 署名を完了した PDf ファイルに対して検証情報を埋め込んだ後に、ドキュメントタイムスタンプを付与する必要がある。ここでは PAdES-Enhanced 署名付与の方法と、検証情報の埋め込み方法を説明する。なお実行にはフル機能の LE:PAdES:Lib が必要。

○ 不可視署名フィールドの作成–PAdES–Enhanced 署名付与（PKCS#12 ファイルの指定）
> LpaCmd -sign -newf -type cades -cert p12 LeTest.p12 test -ts 3161 %TS_URL% s512 ¥
  -in input.pdf -out PAdES1.pdf

※ 上記例の %TS_URL% にはタイムスタンプ局の URL の指定が必要。

○ 検証実行と検証情報の埋め込み
> LpaCmd -verify **-ltv** -in PAdES1.pdf -out PAdES1_ltv.pdf

○ 検証実行と検証情報の埋め込みを分けて実行
> LpaCmd -verify **-value** -result **verify1.xml** -in PAdES1.pdf
> LpaCmd -verify **-ltv verify1.xml** -in PAdES1.pdf -out PAdES1_ltv.pdf

５）PDF 操作

○ 署名付与時に作成日を指定（更新日も同じ日時になる）
> LpaCmd -sign -newf -cert p12 LeTest.p12 test -credate D:20150110132845+09'00' ¥
  -in input.pdf -out credate.pdf

○ 署名付与時に更新日を指定（作成日は変更されない）
> LpaCmd -sign -newf -cert p12 LeTest.p12 test -moddate D:20150110132845+09'00' ¥
  -in input.pdf -out moddate.pdf

○ 署名付与時に XMP ファイルを埋め込む
> LpaCmd -sign -newf -cert p12 LeTest.p12 test -setxmp new.xmp ¥
  -in input.pdf -out signxmp.pdf

○ PDF に XMP ファイルを埋め込む

```
> LpaCmd -pdf -setxmp new.xmp -in input.pdf -out newxmp.pdf
```

○ PDF から XMP ファイルを取得する

```
> LpaCmd -pdf -getxmp save.xmp -in input.pdf
```

※ 実例として sample/LePAdES/cmd の下にあるサンプルソースを参照。

ファイル	概要
CmdAll.bat CmdAll.sh	一括テスト実行
CmdSignTest.bat CmdSignTest.sh	署名サンプル（署名フィールドも同時に作成）
CmdStepTest.bat CmdStepTest.sh	署名サンプル（署名フィールドは別途作成）
CmdDoctsTest.bat CmdDoctsTest.sh	ドキュメントタイムスタンプ サンプル
CmdVerifyTest.bat CmdVerifyTest.sh	検証サンプル
CmdPAdESTest.bat CmdPAdES.sh	長期保管 サンプル（LE:PAdES:Lib が必要）
CmdClientTest.bat CmdClientTest2.bat CmdClientTest.sh CmdClientTest2.sh	クライアント署名サンプル CppClientHexTest.bat はゲートウェイ対策 HEX 通信

コマンド利用のサンプル

３．ＰＤＦ電子署名

３．１．PDF の暗号化と電子署名

PDF の電子署名後に暗号化することは仕組み上できない。逆に暗号化された PDF ファイルに対して権限があれば電子署名を付与することは可能。なお技術的には権限が無くても電子署名を付与することは可能であるが、これは利用者が著作権者であり許諾された場合にのみ利用すべきであり、注意すること。

PDF の暗号化には以下の種類がある。LE:PAdES:Lib ではこのうち「開くパスワード」と「権限パスワード」で暗号化されている場合には、電子署名の付与が可能となっている。「開くパスワード」の場合には引数にパスワード指定が必須となる。

種類		概要
パスワードによるセキュリティ	開くパスワード（ユーザパスワード）	PDF ファイルを開く時に必ずパスワードを要求。 開けるか開けないかのみ制御可能。
	権限パスワード（オーナパスワード）	PDF ファイルを開く時にパスワードは要求しない。 開いた後の編集等の操作に制限を付けられる。 パスワードを入力すると制限は解除される。
証明書によるセキュリティ		証明書の公開鍵で暗号化され秘密鍵を持っている場合のみ利用できる。複数の証明書を指定可能であり、証明書毎に権限パスワードと同じ権限を設定可能。

PDF 暗号化の種類

バージョン	暗号方式	Acrobat 対応	PDF 仕様	補足
V=1	RC4/40bit	Acrobat3.0 以降	PDF1.2	サポート済み
V=2 or V=3	RC4/128bit	Acrobat5.0 以降	PDF1.4	サポート済み
V=4	AES/128bit	Acrobat7.0 以降	PDF1.6	サポート済み
V=5 and R=5	AES/256bit	Acrobat9.0 のみ	PDF1.7/2.0	サポート済み 脆弱性があるので非推奨
V=5 and R=6	AES/256bit	Acrobat-X 以降	PDF1.7/2.0	ISO32000-2 Ver1.07.R1 よりサポート

PDF 暗号化のバージョン対応

LE:PAdES:Lib では現在、未暗号化の PDF ファイルを暗号化する機能はまだ提供していない。必要があれば相談により対応は可能。また証明書によるセキュリティはニーズがほとんど無い為に、現在対応予定はありません。

３．２．署名フィールド

PDF の電子署名は外観を持つが、その為に位置情報（ページ番号と矩形の情報）が必要となる。署名フィールドでは主に電子署名の位置情報を指定する。複数の署名フィールドを区別する為に署名フィールド名が必須となる。LE:PAdES:Lib では PdaField クラスにより情報を指定する。

署名フィールドは電子署名を付与する前に署名する場所だけを決めておくことができる。例えば回覧文書でかつ署名が必要な場合には押印場所のように事前に署名場所を決めておける。署名外観が不要であれば矩形情報に全てゼロを指定することで「不可視署名」にすることも可能。矩形がゼロでは無く範囲がある場合を「可視署名」と呼ぶ。

矩形は X と Y の座標として指定するがページ内のどこを基点とするか指定が可能。PDF 仕様の標準では左下が（0, 0）となっている。LE:PAdES:Lib では、左下・左上・右下・右上から指定が可能。

形式	基点	補足	LpaCmd の指定
LC_LEFT_BOTTOM	左下基点	PDF の座標系と一致	-coordinate LB
LC_LEFT_TOP	左上基点	---	-coordinate LT
LC_RIGHT_BOTTOM	右下基点	---	-coordinate RB
LC_RIGHT_TOP	右上基点	---	-coordinate RT

基点定義（LE_COORDINATE）

３．３．署名辞書

署名辞書は PDF 内部で署名フィールドから関連付けられ、署名データそのものを管理する。改ざんを検知する為の PKI 的要素を管理する最も重要な仕様である。LE:PAdES:Lib では PdaSign クラスにより情報を指定する。署名データの種別により PAdES 仕様では大きく 4 種類に分かれる。LE:PAdES-Basic:Lib では PF_PKCS7_DETACH を、LE:PAdES:Lib では PF_CADES_DETACH を指定する。PF_PKCS7_SHA1 はハッシュ値に署名する方式であり通信量を減らせるメリットがあるのでクライアント署名で利用する。ドキュメントタイムスタンプは単独で利用しても良いし、長期保管時にアーカイブタイムスタンプとしても利用される。

形式	用途	補足
PF_PKCS7_DETACH	PAdES-Basic の署名方式	非推奨（CAdES 形式を利用すべき）
PF_PKCS7_SHA1	PAdES-Basic の特殊用途	クライアント署名 V1 にて利用
PF_CADES_DETACH	PAdES-Enhanced の署名形式	LE:PAdES:Lib のデフォルト署名形式
PF_DOC_TIMESTAMP	ドキュメントタイムスタンプ	正確には署名では無い

署名データ種別（PDA_FILTER_TYPE）

署名の種類には普通署名に対してMDP署名（Modification Detection and Prevention Signature）と呼ばれる署名後の操作を制限する形式がある。MDP 署名を Acrobat では「証明済み署名」と呼んでいる。MDP 署名には署名後に可能な操作の種類によって更に 3 種類に分かれる。なお MDP 署名は署名後の変更を制限できるので便利ではあるが、後から検証情報の埋め込みやドキュメントタイムスタンプを付与する必要がある長期署名（長期保管）には対応できない。長期保管する場合には普通署名を指定する必要がある。

Ver1.03.R2 よりロック署名に対応した。ロック署名は種類としては普通署名であるが、署名後の操作を制限できる。ただしロック署名も MDP 署名と同じ理由（署名後変更の必要あり）で長期署名（長期保管）には対応できない。なおロック署名は正確には署名フィールドに追加される情報である。

形式	説明	長期署名
PS_NORMAL	普通の電子署名（署名を重ねることができる）	○ 可能
PS_MDP_NOCHANG	MDP 署名「変更を許可しない」	× 不可
PS_MDP_FFSIGN	MDP 署名「フォームフィールドの入力と署名フィールドに署名」	
PS_MDP_FFSIGNANNOT	MDP 署名「注釈の作成、フォームフィールドの入力と署名フィールドに署名」	
PS_LOCK_NOCHANG	ロック署名「変更を許可しない」（普通署名）	× 不可
PS_LOCK_FFSIGN	ロック署名「フォームフィールドの入力と署名フィールドに署名」（普通署名）	
PS_LOCK_FFSIGNANNOT	ロック署名「注釈の作成、フォームフィールドの入力と署名フィールドに署名」（普通署名）	

署名種別（PDA_SIGN_TYPE）

なお先にアーカイブタイムスタンプ用の署名フィールドを生成した上で、MDP 署名の PS_MDP_FFSIGN か PS_MDP_FFSIGNANNOT またはロック署名の PS_LOCK_FFSIGN か PS_LOCK_FFSIGNANNOT を指定して署名しておけば、後からアーカイブタイムスタンプの付与も可能であるので、このケースであれば長期署名も可能とは言える。しかしながらアーカイブタイムスタンプ回数が制限される点と、検証情報の埋め込みが MDP 署名とロック署名後に許されるのかと言う点で疑問がある。この為に MDP 署名とロック署名を使った長期署名は推奨しない。

MDP 署名済みまたはロック署名済みの PDF ファイルに対して、新たに署名フィールドの生成をすることはできない。MDP 署名「変更を許可しない」済みまたはロック署名「変更を許可しない」済みの PDF ファイルの既存の署名フィールドに対して、新たに署名を付与することはできない。

３．４．署名外観

署名フィールドの矩形がゼロでは無い「可視署名」の場合には PDF 内部で署名フィールドから関連付けられる署名外観を指定できる。「不可視署名」の場合には署名外観として Blank（何も無い）外観が指定される。LE:PAdES:Lib では PdaAppearace クラスにより情報を指定する。署名外観については PAdES や ISO32000 の仕様には定義されていない。Adobe 社が Acrobat SDK の資料として公開している「Digital Signature Appearances V9 2008-May」に署名外観の情報がある。つまり Acrobat の仕様が事実上の標準となっている。LE:PAdES:Lib においても署名外観は基本的に Acrobat の仕様と互換になるように仕様を決めている。いずれ署名外観についてもきちんと標準化されることを期待しており、標準化された時には対応をする予定である。「Digital Signature Appearances V9 2008-May」は以下よりダウンロードできる。

http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/acrobat_digital_signature_appearances_v9.pdf

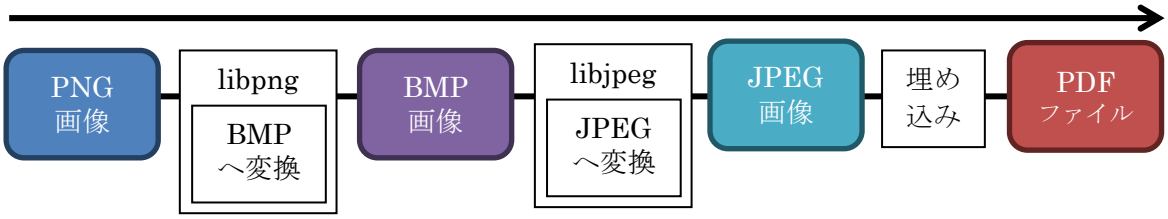
Acrobat の署名外観では署名領域を２分割して、片方をグラフィック表示に、もう片方に情報テキスト表示として使う。どちらか一方をオフにして全面を１つのグラフィック表示にすることも可能となっている。グラフィック部の表示はオフ状態も含めて５種類に対応している。

形式	用途
PG_DESCRIPTION	説明(テキスト)のみ(※グラフィック無し)
PG_NAME	署名者名(署名証明書 CommonName) ※ドキュメントタイムスタンプには不可
PG_GRAPHIC	グラフィックイメージ(※setImage()にて画像ファイル指定が必要)
PG_IMPORTPDF	インポートされた PDF(※setImage()にて PDF ファイル指定が必要)
PG_PNGBITMAP	PNG 画像、透過 PNG 対応(※setImage()にて PNG 画像ファイル指定が必要)

署名外観のグラフィック種別（PDA_GRAPHIC_TYPE）

■ PG_GRAPHIC（画像外観）

グラフィックイメージ（PG_GRAPHIC）には JPEG・BMP・PNG の指定が可能。PDF の内部的には JPEG と同じ形式で埋め込まれ、BMP と PNG からは JPEG 変換している。この為に JPEG 形式を指定すると変換が行われないので画質が保たれる。なお PNG 形式に関しては Ver1.06.R1 からサポートされた PG_PNGBITMAP を使うことで JPEG 圧縮されず画質が保持され、透過 PNG も利用可能になる。この為に PNG 画像に関しては PG_PNGBITMAP の利用を推奨する。

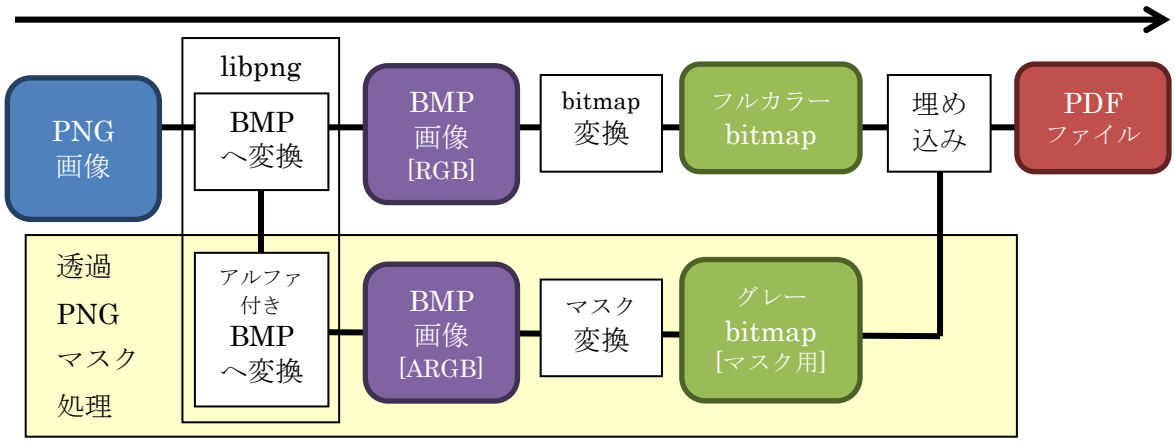


PG_GRAPHIC 指定時の画像の埋め込み処理

■ PG_PNGBITMAP

グラフィックイメージ（PG_ PNGBITMAP）には PNG（透過 PNG）の指定が可能。PDF の内部的にはビットマップを flate 圧縮した形式で埋め込まれるので画質が劣化しない。透過 PNG の場合には別途マスク画像も埋め込まれ、背景を透過したステンシルな署名外観が実現できる。Ver1.06.R1 からサポートされたので、Ver1.05.R1 以前では利用できない。

なお利用可能な等価 PNG の仕様はアルファチャンネル付のフルカラーのみとなる。8bit の透明色を使った PNG には（マスク画像が作成できない為）未対応であるので注意すること。



PG_PNGBITMAP 指定時の画像の埋め込み処理

■ PG_ IMPORTPDF

グラフィックとしてインポート PDF（PG_IMPORTPDF）を使いベクトルを使うと背景を透過としたステンシルな署名外観が利用できる。埋め込みに利用されるのはインポート PDF の最初のグラフィックオブジェクト（XObject）となる。推奨されるインポート PDF の作成方法は、Adobe Illustrator にてベクトル情報を作成して PDF として出力する方法となる。なお Illustrator 上では 1 つのベクトル情報にまとめておく必要がある。

情報テキストを使わない場合には単に PT_NONE を指定する。署名日付の PT_DATE では別途日時フォーマットの指定が可能である。

形式	用途
PT_NONE	テキスト無し(※グラフィックのみ)
PT_NAME	署名者名　証明書の commonName(一般名)
PT_DATE	署名日付　署名辞書の M キー値
PT_REASON	署名理由　署名辞書の Reason キー値
PT_LOCATION	位置情報　署名辞書の Location キー値
PT_SUBJECT	識別名　証明書所有者の X.500 情報文字列
PT_LABEL	ラベル(説明)　PT_NAME～PT_SUBJECT までの説明ラベルの表示
PT_LOGO	背景ロゴ(setLogo()にて PDF ファイル指定が必要)　※現在未サポート
PT_DEFAULT	デフォルト定義（以下フラグをオン状態） PT_NAME、PT_DATE、PT_REASON、PT_LOCATION、PT_SUBJECT、PT_LABEL

署名外観のテキスト設定フラグ（PDA_TEXT_TYPE）

```
/** 署名外観の日時フォーマット指定
 *
 * @param dateFormat [IN] 日時フォーマットを指定する
 * @retval マイナス値 エラーコードが返る
 * @note 未指定時は PDF 標準フォーマット 例 "D:20130530143920+09'00'"
 * @note 指定例 "YYYY/MM/DD hh:mm:ss Z" -> "2013/05/30 14:39:20 +09'00'"
 * @note 年等の桁数変更は不可、24 時間表記のみ
 */
int setDateFormat(const CHAR* dateFormat);
```

日時フォーマットの指定

なおドキュメントタイムスタンプにも署名外観の指定は可能ではあるがテキスト情報は利用できない。従ってグラフィック種別のうち PG_NAME は利用不可であり、テキスト設定では PT_NONE 以外の指定も不可になっている。

３．５．署名検証

署名がある場合には、検証結果に有効・不明・無効の 3 つの状態がある。複数の署名がある場合には、全て VALID なら VALID に、1 つでも INDETERMINATE があり INVALID が無ければ INDETERMINATE に、1 つでも INVALID があれば INVALID になる。

状態	説明	補足
LPK_VS_NONE	署名無し	---
LPK_VS_VALID	有効	検証項目が全て有効であった
LPK_VS_INDETERMINATE	不明（検証情報の不足）	改ざんは検出されないが、CRL/OCSP や認証パスに必要な証明書が足りないか、ルート証明書の信頼性が確認できない等
LPK_VS_INVALID	無効（失効や改ざん等）	1 つでも無効または異常があった

検証結果の状態（LPK_VALID_STATUS）

1 つの署名またはタイムスタンプに付き必要な検証項目は「署名検証 / Signature verification」と「証明書検証 / Certificate validation」に分けられる。両方を合わせて「署名有効性検証 / Signature validation」と呼ぶ。なお署名には署名タイムスタンプが付く場合があるが、署名タイムスタンプについても同様に署名検証と TSA 証明書の証明書検証を行う。

検証項目		説明
署名検証		PDF の署名辞書および署名データにより検証を行う（検証情報は使わない）
	署名値の確認	署名対象に対して署名証明書の公開鍵で検証をして改ざん有無を確認
	必須項目の確認	仕様に定められている必須の属性や PDF 要素が含まれている事を確認 仕様で禁止されている属性や PDF 要素が含まれていない事を確認
証明書検証		検証情報（証明書/CRL/OCSP）を集めて署名時刻/現在時刻で検証を行う ※ 検証情報が不足している場合には INDETERMINATE になる
	認証パスの構築	ルート証明書（自己署名証明書）まで認証パスを構築できる事を確認
	認証パスの失効確認	認証パスに含まれる全ての証明書の失効を確認 ※ PAdES の検証情報（valid data）には利用優先順位があるので従う ※ 最後のルート証明書は通常失効確認はされない
	トラストアンカー確認	認証パス最後のルート証明書が信頼済みのトラストアンカーか確認 ※ 独自証明書ストアの trusts にあるか、Windows 証明書ストアで確認

検証項目（検証結果レポートされる項目）

証明書検証では失効の有無を、署名時刻（署名タイムスタンプ時刻か署名後最初のタイムスタンプ時刻）か、署名時刻が確認できない場合には現在時刻において確認する。現時点で失効していても署名時刻において有効であれば問題なく有効である。現在 PAdES の検証仕様が JNSA にて策定中であり、PAdES 標準検証仕様が確定した段階で PAdES 標準検証仕様に従った検証に対応する予定である。

CRL/OCSP には署名が付いているがルート証明書の移行があった場合には、PDF 署名の署名証明書のルート証明書と、CRL/OCSP 署名の署名証明書のルート証明書が異なる場合がある（通常は一致する）。LE:PAdES:Lib では、両方のルート証明書がトラストアンカーと信頼されておりかつ発行者名が同じ場合にはエラーにならない仕様となっている。発行者名が異なるかトラストアンカーとして確認できない場合には不明（INDETERMINATE）となる。

PAdES の長期保管の一般的な形式は署名タイムスタンプ付き署名の後で検証情報（DSS/VRI）を埋め込み最後にドキュメントタイムスタンプを付与した形になる。検証結果の例を以下に示す。

```
LE:PAdES:Lib Signature Verify Result Report V1.02.R2

署名有効性検証: 有効(VALID) - 署名は改ざんされておらず証明書も有効

 文書タイムスタンプ(DocTS): 名前(DocTS1) ハッシュ方式(sha512)
       署名データハッシュ(2149EA775C1820CD0F47C1C1879B4801F5859689)
       ByteRange [0 49410 69892 509]
  署名検証: 有効(VALID) - 署名は改ざんされておらず有効
         署名時刻: 201310190054l7Z
  証明書検証: 有効(VALID) - 証明書は有効
   証明書: 検証(有効(VALID)) 取得(data) 検証(CRL)
          名前(LE TSA 200001)
          番号(200001) 期限(20230829000000Z)
    CRL: 取得(network) 更新(20131018190243Z)
   証明書: 検証(有効(VALID)) 取得(data)
          名前(LangEdge CA Root 2)
          番号(038D7EA4C68002) 期限(20180420015046Z)

 署名: 名前(SIGN2) 種類(PAdES-Enhanced) ハッシュ方式(sha256)
       署名データハッシュ(70388D171F54CA1A639BA819790FF21E54CC88A3)
       ByteRange [0 10941 31423 11985]
  署名検証: 有効(VALID) - 署名は改ざんされておらず有効
         署名時刻: D:20130925095734+09'00'
  証明書検証: 有効(VALID) - 証明書は有効
   証明書: 検証(有効(VALID)) 取得(data) 検証(CRL)
          名前(LE Test 100001)
          番号(100001) 期限(20160415090000Z)
    CRL: 取得(pdfDss) 更新(20131017190243Z)
   証明書: 検証(有効(VALID)) 取得(data)
          名前(LangEdge CA Root 1)
          番号(038D7EA4C68001) 期限(20180420015025Z)
  署名タイムスタンプ(SigTS): ハッシュ方式(sha512)
   署名検証: 有効(VALID) - 署名は改ざんされておらず有効
          署名時刻: 201309250057l5Z
   証明書検証: 有効(VALID) - 証明書は有効
    証明書: 検証(有効(VALID)) 取得(data) 検証(CRL)
           名前(LE TSA 200001)
           番号(200001) 期限(20230829000000Z)
     CRL: 取得(pdfDss) 更新(20131017190243Z)
    証明書: 検証(有効(VALID)) 取得(data)
           名前(LangEdge CA Root 2)
           番号(038D7EA4C68002) 期限(20180420015046Z)
```

最終的な検証結果

ドキュメントタイムスタンプ

署名

署名タイムスタンプ

署名（署名タイムスタンプ付）

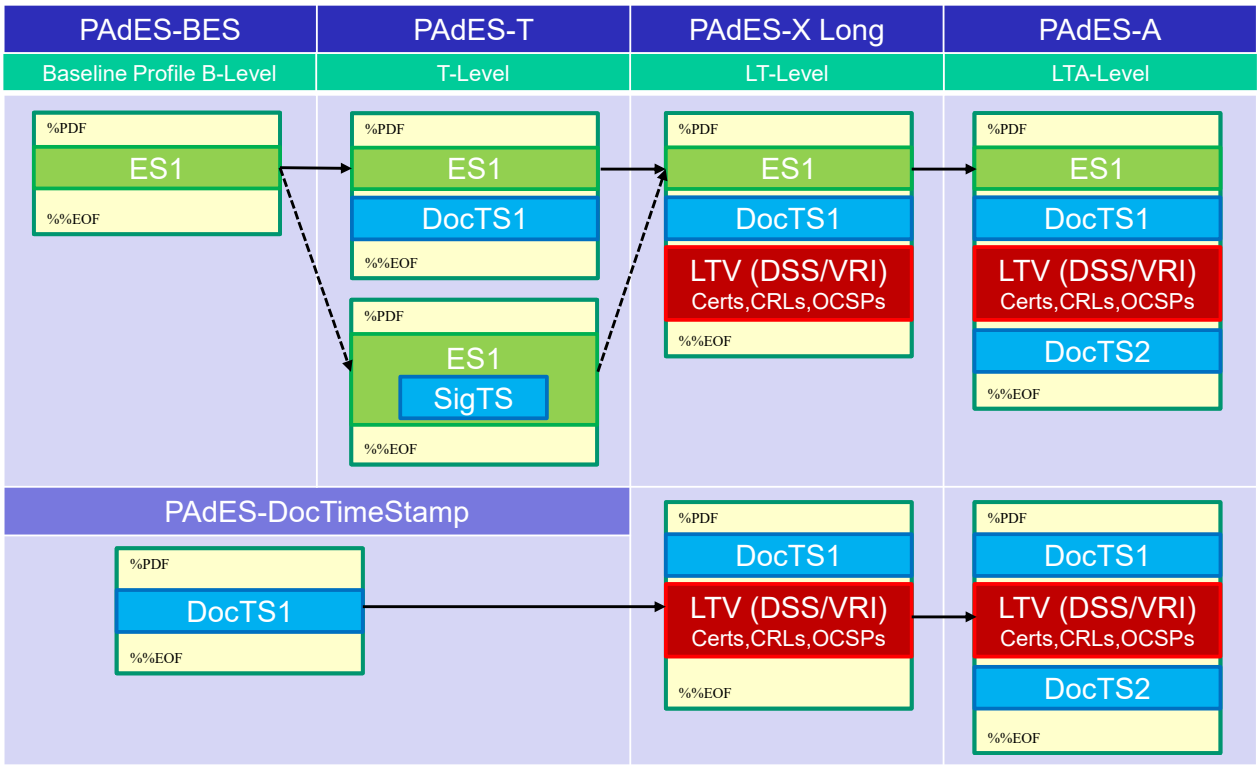
署名（署名タイムスタンプ付）＋ドキュメントタイムスタンプの検証結果例

検証情報の取得先の一覧を以下に示す。検証結果では PKI 要素はどこから取得したかを以下の表示文字列で示している。

LPK_INFO_FROM 定義	表示文字列	説明
LPK_FM_UNKNOWN	unknown	取得先不明（エラー）
LPK_FM_DATA	data	署名データから取得
LPK_FM_ORGSTORE	orgStore	独自証明書ストアから取得
LPK_FM_ADDSTORE	addStore	追加した検証情報から取得、CVS 取得情報等
LPK_FM_WINSTORE	winStore	Windows 証明書ストアから取得
LPK_FM_NETWORK	network	ネットワークから取得
LPK_FM_VALID	valid	検証データ（OCSP/CRL 等）から取得
LPK_FM_PDF_DSS	pdfDss	PAdES：PDF の DSS 辞書から取得
LPK_FM_NOTUSE	notUse	未利用（ネットワーク接続しない時等）
LPK_FM_LCACHE	cache	キャッシュから取得
LPK_FM_TS_DATA	tstData	XAdES：TimeStampToken から取得
LPK_FM_KEYINFO	keyInfo	XAdES：KeyInfo から取得
LPK_FM_SIGN_VALID	sigValid	XAdES：CertificateValues/RevocationValues から取得
LPK_FM_SIGTS_VALID	stsValid	XAdES：署名タイムスタンプの TimeStampValidationData から取得
LPK_FM_ARCTS_VALID	atsValid	XAdES：アーカイブタイムスタンプの TimeStampValidationData から取得

３．６．長期署名の運用

PAdES も長期署名（AdES）の一種であり、XAdES/CAdES と同じように署名レベルがある。通常は ES(ES-BES/EPES) → ES-T → ES-X Long → ES-A と情報を追加して行く。PAdES の場合には ES/ES-T だけでは無く DocTimeStamp のみ（Content TypeStamp）と言う仕様もある。また ES-T に関しては署名データへの埋め込み署名タイムスタンプ（SigTS）だけでは無く、署名データの後に DocTimeStamp（DocTS）を追加しても良い。



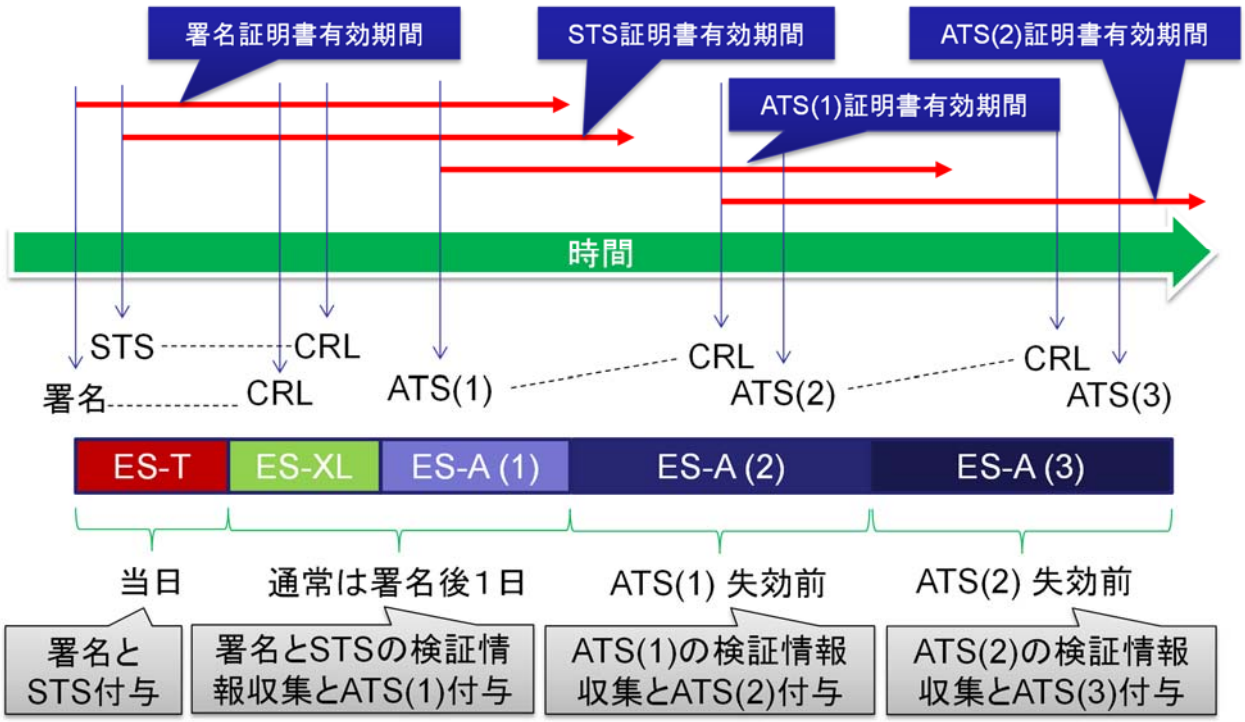
PAdES の署名レベル（図の左から右へ更新して行く）

署名レベルを Adobe 社の Acrobat や Reader で判別するのは現在非常に難しい。署名パネルを表示して ES / SigTS / DocTS / LTV の構成を確認する必要がある。

複数の署名（ES）がある場合も判断が難しい。明確な仕様定義は無いが署名毎に個別に必要要素を確認して署名レベルを判断する必要がある。また DocTimeStamp のみ（Content TypeStamp）も明確な仕様がまだ定まっていない。この辺りは今後 JIS プロファイル等で考慮する方向で検討が進んでいる。

なお署名レベルや複数署名に関しては LE:PAdES:Lib では判定機能を将来追加する予定である。

検証情報（LTV）の埋め込みに CRL を使う場合には失効情報の猶予期間を考慮する必要がある。その為に PAdES-T までは一気に作成しても構わないが、猶予期間を考慮するなら PAdES-X Long にするには CRL の更新を待つ。通常 CRL の場合には 1 日待つと更新されるポリシが多いが利用する認証局に確認が必要となる。OCSP はリアルタイムに失効情報が取得できる場合もあるが、やはり情報更新に時間がかかるケースも多いのでこれも確認が必要となるが、通常は猶予期間を考えずすぐに PAdES-X Long 化しても良い。PAdES-A を生成した後更に延長する場合には PAdES-X Long と PAdES-A を繰り返して追加して行く。なお Ver1.08.R1 からは OCSP 優先になった。CRL と OCSP の優先について詳しくは「LE:PKI:Lib マニュアル」（LePKI-manual.pdf）の「３．１１．失効情報取得の高度な指定」を参照。



猶予期間を考慮した長期署名生成のタイミング例（CRL が毎日更新される場合）

なお検証情報は埋め込み（LTV）以外に、外部から与える事もできるし、ネットワークから現在の情報を取得する事も出来る。PAdES の仕様では利用する優先順位が以下のように決められている。

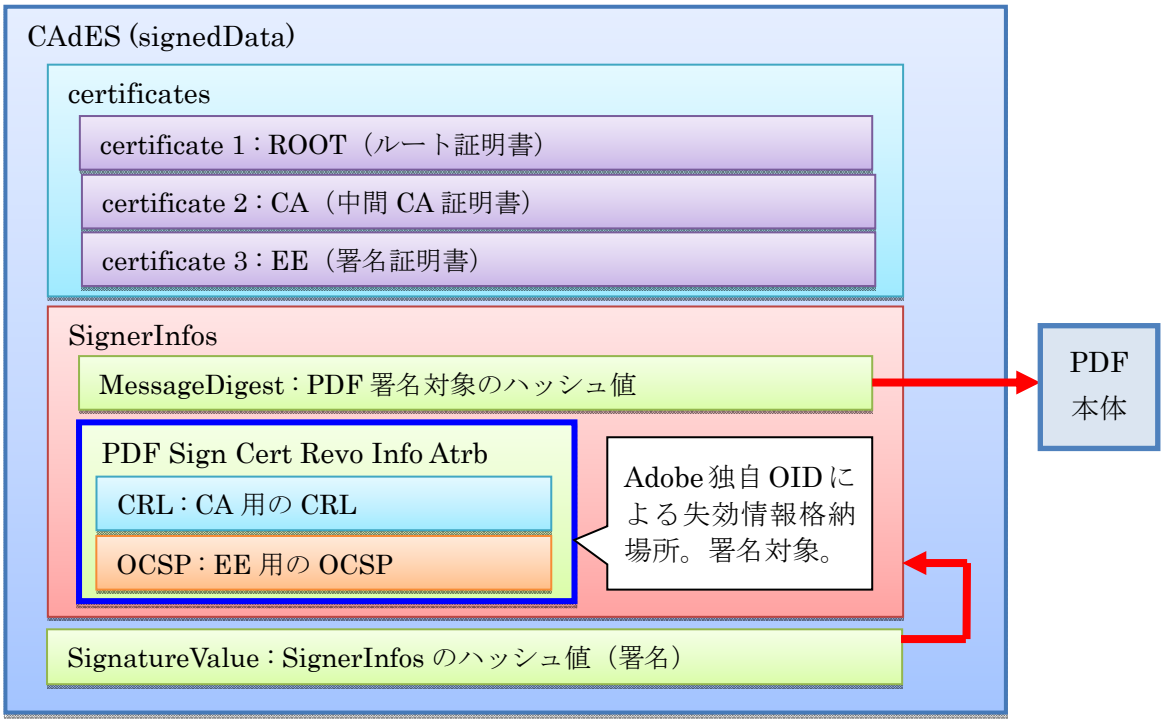
優先順位	検証情報の格納場所
1	DSS の中の署名 VRI 要素から参照されている検証情報
2	DSS から参照されている検証情報
3	署名データ自身に埋め込まれている検証情報
4	ローカルなリポジトリに保管されている検証情報（外部から提供）
5	オンラインソースから取得された検証情報（ネットワークから取得）

PAdES 仕様における検証情報の利用優先順位

３．７．Adobe 仕様の失効情報（V1.06.R1 以降）

CAdES を使った署名データには Adobe 独自の PDF Sign Cert Revo Info Atrb により失効情報を署名前に埋め込んでおくことが可能な仕様がある。なお本仕様は PAdES 仕様にも記載されているので PAdES の範囲内の仕様であり、Adobe 製品ではサポートされている。V1.06.R1 よりこの Adobe 独自の失効情報の埋め込みと検証に対応した。

PDF Sign Cert Revo Info Atrb により埋め込まれる失効情報（CRL/OCSP）は署名対象（SignerInfos）に含まれる為に、署名前に埋め込む必要がある。以下に本仕様を使った CAdES 署名データのサンプルを示す。



PDF Sign Cert Revo Info Atrb を使った CAdES 内部構造例

PDF Sign Cert Revo Info Atrb を使うには前提条件がある。条件のうち幾つかは認証局（CA）に依存するものであり、安易な利用は推奨しないので必ず相談すること。

利点	署名時に失効情報の埋め込みが完了できる。（二度手間が無い）
	Adobe 製品にて検証可能である。（Adobe 独自仕様）
欠点	失効情報が大きくなると署名データに入らないのでエラーになる。 → 署名データ部を大き目に指定しておき OCSP 利用が前提となる。
	署名前の情報になるので猶予期間は考慮できない。 → HSM 利用（リモート署名）のように秘密鍵が漏えいしない環境等が前提となる。
	未サポートの実装や製品も多いと予想される。

PDF Sign Cert Revo Info Atrb を使う場合の利点と欠点

V1.06.R1 以降では PDF Sign Cert Revo Info Atrb の失効情報は検証時には自動的に判別して利用される。

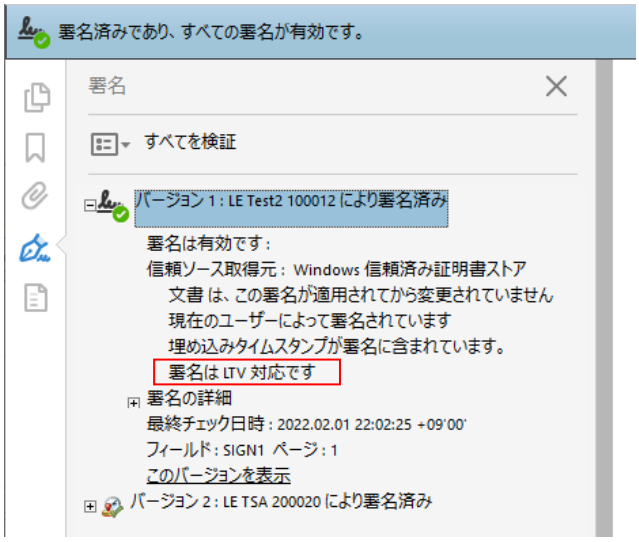
V1.06.R1 以降では CAdES 署名（LePAdES::addEnhanced）時の PdaSign クラスで設定する署名フラグに（PDA_SIGN_FLAG::PF_SIGN_ADDREVO）を指定することで、署名時に失効情報を取得して PDF Sign Cert Revo Info Atrb として埋め込みを行う。しかしながら前述したように本オプションフラグの利用には前提となる環境が必要となるので注意が必要である。なお LpaCmd では署名時の -opt 引数に arevo を指定する。

```
-opt <none/nochn|prevf|arevo> : 署名オプション 省略時=none（docts では無効）
   none  : 署名オプション無し
   nochn : 署名証明書のチェーンを埋め込まない（非推奨）
   prevf : 署名前に署名証明書の検証を行う（検証に成功しないと署名できない）
   arevo : 証明書の検証を行い検証情報を署名対象に追加する（cades のみ）
```

※ **V1.07.R2 以降では PKCS#7 形式の署名検証にも PDF Sign Cert Revo Info Atrb の失効情報を利用して検証できるようになった。ただし PKCS#7 の署名データに PDF Sign Cert Revo Info Atrb の失効情報を埋め込む機能は提供されていない。**

３．８．長期検証（LTV）対応の判定（V1.07.R3 以降）

長期検証状態とは LTV（Long-Term Validation）と呼ばれ検証に必要な検証情報（証明書/CRL/OCSP）が全て PDF の中に格納されている状態を示す。LTV 対応の PDF は検証時に外部にネットワーク接続しないで検証が可能となる。Adobe Reader/Acrobat の検証パネルでも LTV 対応かどうかを表示している。



Adobe Acrobat による LTV 対応の表示例

LePAdES の検証結果 XML では証明書/CRL/OCSP を検証時にどこから取得したかを示す from 属性を持っているが、LTV 対応の場合にはネットワーク接続して取得した from="network" が 1 つも無いことになる。V1.07.R3 よりこの判定を行う PdaVerifyXml.isLTV() が追加された。検証結果 XML を PdaVerifyXml クラスにセットした上で isLTV() により LTV 対応なら true を返す。

```
/** LTV(長期検証状態)を判定して返す
 *
 * @retval LTV の場合=1, LTV では無い場合=0
 * @retval マイナス値 エラーコードが返る
 */
int isLTV() const;
```

PdaVerifyXml クラスの LTV 判定 API

LpaCmd コマンドを使った場合にも検証引数に LTV 判定する -isltv を追加した。

-isltv　　：検証結果が LTV かどうかを表示

３．９．署名操作時の XMP 更新（V1.07.R3 以降）

署名やタイムスタンプを付与して PDF 保存をする際に、PDF の文書情報辞書の更新日（ModDate キー）を更新すると同時に XMP 情報がある場合には XMP の更新日（ModifyDate タグ要素）も更新している。この為に PDF 読み込み時には XMP の解析を行っている。しかし PDF としては正しいが埋め込まれている XMP の仕様が不正で読み込めない場合がある。不正のケースとしては、XML として不正な文字が使われていたり、XMP として認められないタグ要素が使われていることがあった。

V1.07.R3 より前のバージョンではこのような不正な XMP を持った PDF ファイルには署名やタイムスタンプを付与することが出来ない問題があったが、V1.07.R3 では PDF 読み込み時に XMP の解析をスキップする LePAdES.loadXmp(bool load) がサポートされた。PDF 読み込みの API である LePAdES.loadFile()/loadStream()/loadBinary() を呼び出す前に LePAdES.loadXmp(false) を指定することで XMP 読み込みを行わず署名の付与等を行うことが出来る。ただし loadXmp(false) を指定した場合には XMP の更新日（ModifyDate タグ要素）は更新されない。この状態でも署名済みの PDF としては何も問題は無い。

```
/** XMP 読み込み設定¥n¥n
 *
 * @param load [IN] XMP 読み込みしない場合は false を指定する
 * @note 未指定時のデフォルト設定は true となる
 * @note loadFile/loadStream/loadBinary を呼び出す前に利用する
 * @note XMP 異常(-2054)があるが PDF ファイルとしては正常な場合に false を指定する
 * @note false を設定した場合には XMP に関する操作は行うことが出来ない
 */
 void   loadXmp(bool load);
```

C++の XMP 読み込み API

LpaCmd コマンドを使った場合にも -noxmp 引数を使って XMP 読み込みをスキップできるようになった。

-noxmp　　：XMP 読み込みしない(XMP 操作しない)

４．ＰＫＩ要素と電子署名付与

署名時の証明書（LpkCert）の利用方法やタイムスタンプ（LpkTimestamp）の使い方や独自証明書ストア等の、PKI 要素に関しては「ラング・エッジ PKI 基本ライブラリ LE:PKI:Lib マニュアル」（LePKI-manual.pdf）を参照。本書では重複するが独自証明書ストアは重要であるので記載する。

４．１．独自証明書ストア

一般に Windows 環境では「Windows 証明書ストア」が、Java 環境では「Java 証明書ストア」が利用される。LE:PAdES:Lib は C++ にて開発されている関係で「Java 証明書ストア」が利用できない。その為に LE:PAdES:Lib では「独自証明書ストア」を提供する。なお Windows 版では「Windows 証明書ストア」も利用が可能である。

「独自証明書ストア」は「信頼済みルート証明書（trusts）」「中間証明書（certs）」「検証情報（valids）」のディレクトリで構成される。LePKI::setStore() でディレクトリ指定と読み込みを行う。ディレクトリ指定を NULL にすると実行ファイルのあるディレクトリ下の store フォルダが指定される。例えば LpaCmd コマンドであれば、LpaCmd の下にあるディレクトリとなる。サーバ組み込み時には明示的なディレクトリの指定を推奨する。その他 addTrust() / addCert() / addCrl() / addOcsp() の各 API を使って独自証明書ストアに情報追加も可能である。なお署名証明書（と秘密鍵）は PKCS#12 形式のファイルで指定するので独自証明書ストアには含まれない。

フォルダ・ファイル	概要
LpaCmd	実行ファイル（Windows は LpaCmd.exe）
store	証明書ストアディレクトリ（setStore()で指定可能）
trusts	信頼済みルート証明書（トラストアンカー証明書）用ディレクトリ
*.cer / *.der	証明書群（拡張子 .cer と .der を読み込む）
certs	中間証明書用ディレクトリ
*.cer / *.der	証明書群（拡張子 .cer と .der を読み込む）
valids	失効情報（CRL/OCSP）用ディレクトリ
*.crl / *.ocsp	失効情報群（拡張子 .crl と .ocsp を読み込む）

独自証明書ストアのディレクトリ構成

LE:PAdES:Lib の Windows 版では Windows 証明書ストアから、信頼済みルート証明書（"ROOT"）と中間証明書（"CA"）を取得する。setStoreFlag(FLAG flag) の引数として LPK_WIN_STORE をオフにして指定すると Windows 証明書ストアは利用されない。

４．２．署名鍵の置き場所による分類

電子署名システムを設計する上で最も重要な点は署名に使う（署名値の計算を行っている）署名鍵（秘密鍵）をどこに置くかを決めることである。LePKI ライブラリでは PKCS#12 ファイル形式の署名鍵/証明書をサポートしているが、署名鍵はコピーや漏洩をしない HSM（ハードウェア セキュリティ モジュール）や IC カード/USB トークン等に格納して運用されることが望ましい。なお電子証明書（公開鍵）は通常秘匿されるものではない（JPKI のマイナンバーカード等が例外）ので、X.509 ファイル形式で保存して利用して構わない。

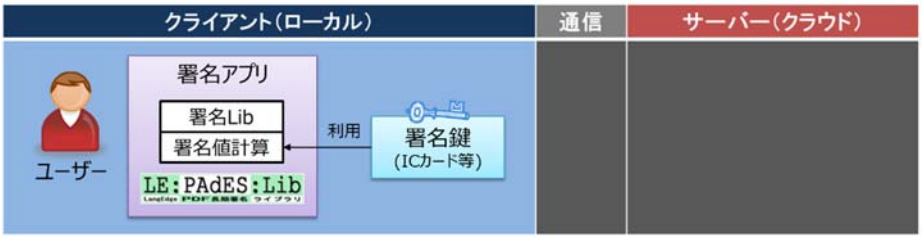
署名鍵の保持には「ユーザー所有」「サーバー保管」「外部サービス」の 3 パターンがある。署名方式毎に LE:PAdES:Lib の使い方を以下にまとめる。「ユーザー所有」の場合には「ローカル署名」と「クライアント署名」に分かれる。また「サーバー保管」の場合には PKCS#12 ファイルか HSM/クラウド HSM かによって実装方法が分かれる。「外部サービス」の場合にはリモート署名のデファクト標準である CSC API が良く使われている。（CSC：Cloud Signature Consortium）

LE:PAdES:Lib には、署名値を直接自分で計算する「直接署名」の組み込み方と、署名値を外部で計算する為の「仮署名」の組み込み方の 2 種類がある。署名方式を決めてどちらを利用するか決める必要がある。

署名鍵	署名方式		概要
ユーザー所有	ローカル署名		LE:PAdES:Lib をネイティブアプリに組み込んで利用。 ※　直接署名 API：LePAdES.signXXX() 利用
	クライアント署名 (LE:Client:Sign)		LE:PAdES:Lib は署名サーバーに組み込む。 署名値の計算のみローカル PC で行う。 クライアント側に LE:Client:Sign を組込み連携する。 ローカル PC に LE:Client:Sign インストールが必要。 LE:PAdES:Lib のクライアント署名連携 API を利用可。 ・　**クライアント署名連携部の追加開発が必要。** ※　仮署名 API：LePAdES.makeXXX() 利用
サーバー保管	サーバー署名	PKCS#12 安全性低	LE:PAdES:Lib は署名サーバーに組み込んで利用。 ※　直接署名 API：LePAdES.signXXX() 利用
		HSM 安全性高	LE:PAdES:Lib は署名サーバーに組み込む。 署名値の計算のみ HSM/クラウド HSM で行う。 ・　**HSM/クラウド HSM 連携部の追加開発が必要。** ※　仮署名 API：LePAdES.makeXXX() 利用
外部サービス (RSSP)	リモート署名		LE:PAdES:Lib は署名サーバーに組み込む。 署名値の計算のみリモート署名サービスで行う。 ・　**リモート署名サービス連携部の追加開発が必要。** ※　仮署名 API：LePAdES.makeXXX() 利用

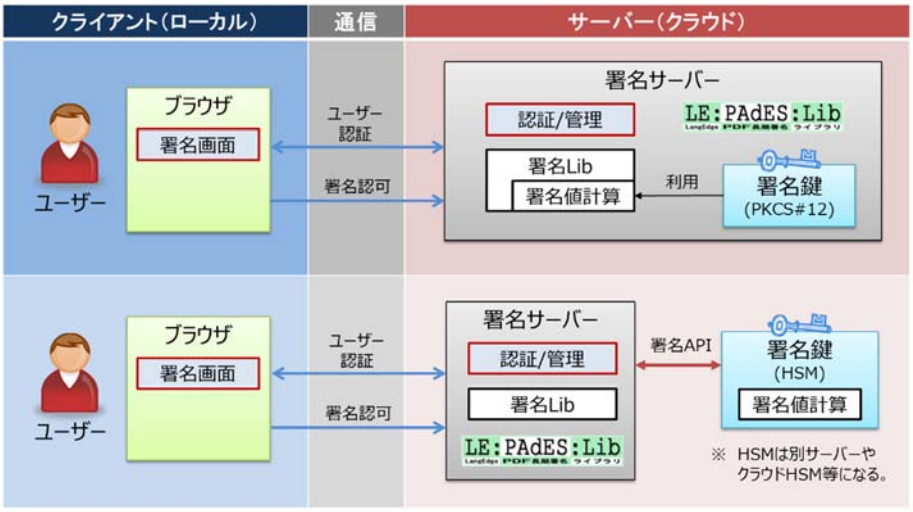
署名鍵の保持方法による電子署名システムの分類

A）ローカル署名方式

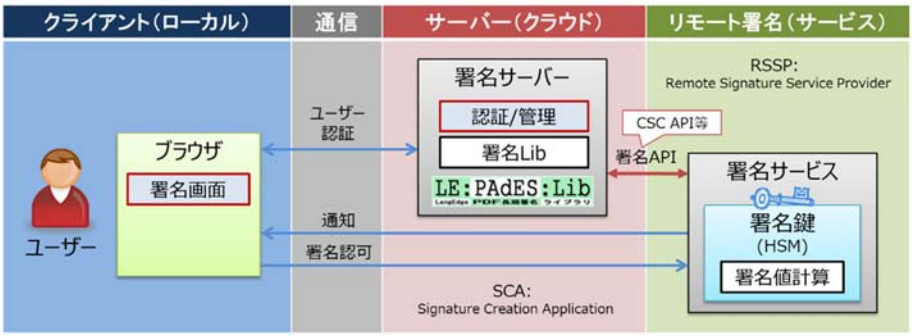


B）クライアント署名方式

C）サーバー署名方式（上：PKCS#12 ファイル利用／下：HSM かクラウド HSM 利用）



D）リモート署名方式



４．３．仮署名（HSM の利用等）

サーバー上で署名を行うサーバー署名の仕組みでは別ハードウェア上で署名値計算を行う HSM の利用が推奨される。署名値のみを外部で計算すると言う意味ではクライアント署名（７．クライアント署名 参照）と同じではあるが、サーバー側に置かれる HSM またはクラウド HSM にて署名値の計算を行う点で異なる。外部で計算された署名値を使う為には**仮署名**（LE:PAdES:Lib の造語：外部で署名値を計算する為のハッシュ値計算と途中結果の PDF 取得を行う）を使う必要がある。

HSM を使った場合の署名の処理手順を以下に示す。「API 実行 2」は V1.07.R1 からサポートされた LePAdES.makeEnhanced2() を利用した手順であり、「API 実行 1」は V1.07.R1 より前からサポートしていた LePAdES.makeEnhanced() を利用した手順である。makeEnhanced2 では LpkCades や LpkCrypto を使う処理を内部的に行ってくれる為に、LePAdES クラスのみで処理が行える点と、利用する API が少なくなっている（シンプルになっている）と言う利点がある。ただし署名タイムスタンプを追加する為には LePAdES.getContents() により CAdES データを取得して、LpkCades クラスとして addTimestamp() を実行する必要がある。

処理手順		API 実行 1	API 実行 2（V1.07.R1 以降）
LE:PAdES:Lib の処理（仮署名）			
1	PDF 仮署名の実行	LePAdES.makeEnhanced()	LePAdES.makeEnhanced2() ※ CAdES 生成まで行う
2	PDF 仮署名済み PDF 保存かリロード	LePAdES.saveFile()	LePAdES.saveFile()
3	PDF 署名対象（ByteRange）の取得	LePAdES.getTarget()	---
4	PDF 署名対象のハッシュ値計算	LpkCrypto.getHash()	---
5	CAdES 仮署名の実行	LpkCades.sign()	---
6	CAdES 署名対象のハッシュ値取得	LpkCades.getSignTarget()	※ makeEnhanced2 で取得
HSM の処理（署名値の計算）			
7	ハッシュ値から署名値の計算	※ PKCS#11 や Java SPI や RESTful API 等で計算	
LE:PAdES:Lib の処理（署名値埋め込み）			
8	CAdES に署名値を埋め込み	LpkCades. setSignValue()	LePAdES. setSignValue()
9	署名済み CAdES をバイナリ取得	LpkCades.getBin()	---
10	CAdES バイナリを PDF に埋め込み	LePAdES.setContents()	---
11	PDF 署名済み PDF 保存	LePAdES.saveFile()	LePAdES.saveFile()
LE:PAdES:Lib の処理（オプション：署名タイムスタンプの追加）			
A	CAdES の取得	---	LePAdES.getContents()
B	CAdES のセット	---	LpkCades.setBin()
C	CAdES へタイムスタンプ追加	LpkCades.addTimestamp() ※ 手順 8 と 9 の間で実行	LpkCades.addTimestamp()
D	CAdES バイナリを PDF に埋め込み	---	LePAdES.setContents()
E	署名タイムスタンプ済み PDF 保存	---	LePAdES.saveFile()

Java 用の LePAdES.makeEnhanced2() を使った実装サンプルが sample/LePAdES/java/LePAdES_hsm.java にある。

V1.07.R2 より LpaCmd コマンドを使った仮署名も行えるようになった。-sign 時の引数として、-make2/-value2/-setval2 が追加された。実装サンプルが sample/LePAdES/cmd の下に、CmdHsmTest.bat / CmdHsmTest.sh としてある。

-make2	：仮署名 2 (CAdES 署名データを生成しハッシュ値(HEX)を返す)
-value2 hash	：仮署名 2 (試験用：ハッシュ値から署名値を計算)
-setval2 val	：仮署名 2 署名値埋め込み (make2 済みファイルに署名値埋め込み)

※ 仮署名コマンド実行イメージ

```
// 仮署名 2 の実行（test.cer と test.p12 は同じ証明書）
LpaCmd -sign -make2 -newf -name SIGN1 -cert x509 test.cer -in in.pdf -out temp.pdf > HASH_VALUE
// 後処理：変数 $HASH_VALUE に実行結果をセット
// ハッシュ値から署名値の計算（HSM の代わり）
LpaCmd -sign -value2 $HASH_VALUE -cert p12 test.p12 test > SIGN_VALUE
// 後処理：変数 $SIGN_VALUE に実行結果をセット
// 署名値の埋め込み
LpaCmd -sign -setval2 $SIGN_VALUE -name SIGN1 -in temp.pdf -out signed.pdf
```

HSM/クラウド HSM 用に認証局から電子証明書を発行してもらう為には、HSM 上に鍵ペアを生成して認証局から指定された形式の CSR（Certificate Signing Request：証明書発行要求）ファイルに署名鍵で署名して提出する必要がある。鍵ペアの生成方法や CSR ファイルの取得方法等は HSM 毎に異なるケースが多く、HSM を提供しているベンダーやサービスにやり方等を確認する必要がある。

現在 HSM の利用は可能ではあるが電子証明書の発行も含め比較的煩雑な手順が必要であるので、可能であれば別途サポートを受けることを推奨する。また HSM の利用方法はベンダーやサービス毎に異なるのでその点でも別途のサポートが必要となるケースは多い。

４．４．証明書検証サーバー（CVS）の利用：GPKI/LGPKI 等

Ver1.05.R1 より GPKI（日本政府 PKI）や LGPKI（地方自治体 PKI）で利用されている、証明書検証サーバ（CVS）の利用が可能となった。CVS を利用した場合には署名証明書の認証パスの確認等の検証は全てサーバー側で行える。この為に CVS の利用は別途実装が必要となる。API は LpkUtil::getCvs() となる。なお CVS 応答は、基本的に OCSP 応答の拡張となっている。

```
/** CVS の取得(V1.08.R1 以降)¥n¥n
 *
 * @param cvs [OUT] 取得した cvs バイナリ
 * @param cert [IN] 失効確認の対象となる証明書を指定
 * @param parent [IN] 失効確認の対象となる証明書の親証明書を指定
 * @param opt [IN] GPKI 拡張の応答フォーマットをセット（value = 1 でセット）
 * @param url [IN] CVS 取得 URL(http のみ指定可能)
 * @param basic [IN] 基本認証用指定(空白文字で区切り"ID PASSWORD"で指定)
 * @param htype [IN] HTTP 通信方式を指定(Windows のみ指定可能、Linux は LPK_HTTP_SOCKET 固定)
 * @retval マイナス値 エラーコードが返る
 */
int getCvs(
        BINARY& cvs,                    // 結果
        const LpkCert& cert,            // 検証対象の証明書
        const LpkCert& parent,          // 通常 GPKI または LGPKI のルート証明書
        FLAG opt,                       // 1 をセットすると証明書群/CRL 群/OCSP 群を返す
        const CHAR* url,                // CVS サーバーの URL
        const CHAR* basic=NULL,         // Basic 認証情報（通常指定不要）
        LPK_HTTP_TYPE htype=LPK_HTTP_DEFALT     // HTTP 通信方式指定（通常指定不要）
);
```

C++の証明書検証サーバー利用 API：LpkUtil クラス

取得した CVS のバイナリイメージは LpkOcsp クラスの setBin() によりセットして結果の取得が可能となっている。

```
/** GPKI 拡張の認証パスステータスを取得（無い場合はマイナス値が返る）
 *
 * @param num [IN] 複数ある場合に情報番号を指定（通常 0 で良い）
 * @retval int GPKI 拡張の認証パスステータスが返る
 */
int getCertPathStatus (int num = 0) const;

/** 認証パスの証明書を取得(GPKI)
 *
 * @param num [IN] 複数ある場合に情報番号を指定（通常 0 で良い）
 * @retval LpkCerts 認証パスの証明書群が返る
 */
LpkCerts getCertPath (int num = 0) const;

/** CRL/ARL を取得(GPKI)
 *
 * @param num [IN] 複数ある場合に情報番号を指定（通常 0 で良い）
 * @retval LpkCrls 認証パスに必要な CRL/ARL 群が返る
 */
LpkCrls getRevocationList (int num = 0) const;

/** OCSP レスポンスを取得(GPKI)
 *
 * @param num [IN] 複数ある場合に情報番号を指定（通常 0 で良い）
 * @retval LpkOcsps 認証パスに必要な OCSP 群が返る
 */
LpkOcsps getOCSPResponse (int num = 0) const;
```

C++の LpkOcsp クラスの GPKI 関連 API

```
/** トラストアンカーを指定して署名検証を行い結果を XML 形式で返す(V1.08.R1 以降)¥n¥n
 *
 * @param xml [OUT] 署名検証結果 XML が返される
 * @param value [IN] true なら証明書や検証情報を XML に埋め込む（addLTV()に時は true が必須）
 * @param fieldName [IN] 対象となる署名フィールドを指定、NULL なら全署名が対象となる
 *@param verifyDate [IN] 検証時刻を GeneralizedTime 型で指定(試験用なので通常は NULL を指定)
 * @param rootCert [IN] 必要な場合に署名証明書のトラストアンカーを指定(通常は null で良い)
 * @param flag [IN] 検証フラグ(PDA_VERIFY_FLAG)を指定(addLTV()に使う時は PVF_NONE が必須)
 * @retval マイナス値 エラーコードが返る
 * @note 出力された XML は PdaVerifyXml クラスで読み込んで情報取得する
 * @note 出力された XML は addLTV()の指定にも使われる
*/
int verify(
        BINARY& xml,                    // 結果
        bool value,                     // LTV 化時には true を指定
        const CHAR* fieldName,          // 署名フィールド名指定
        const CHAR* verifyDate,         // 検証時刻（通常 NULL で良い）
        LpkCert* rootCert,              // 通常 GPKI または LGPKI のルート証明書
        FLAG flag = PVF_NONE            // PVF_NONE を指定
);
```

C++の証明書検証サーバー結果を利用した検証 API：LePAdES クラス

証明書検証サーバー（CVS）を使った LTV 化の手順説明：

手順	利用 API	処理内容
Step.1	LpkUtil::getCvs()	CVS の URL とトラストアンカー（BCA ルート）証明書と value=1 を指定して getCvs を実行する。
Step.2	LpkOcsp::setBin()	取得した CVS バイナリを LpkOcsp クラスにセットする。
Step.3	LpkOcsp:: getCertPathStatus()	取得した CVS 応答のステータスを取得してチェックする。 ステータスが 0 以外の場合はエラーなので以下手順は行えない。 ※　詳細は GPKI や LGPKI の技術仕様書を参照。
Step.4	LpkOcsp:: getCertPath()	取得した CVS 応答から証明書群 LpkCerts クラスを取得する。
Step.5	LePKI::addCert()	全取得証明書をループして順番に独自証明書ストアに追加する。
Step.6	LpkOcsp:: getRevocationList()	取得した CVS 応答から CRL 群 LpkCrls クラスを取得する。
Step.7	LePKI::addCrl()	全取得 CRL をループして順番に独自証明書ストアに追加する。
Step.8	LpkOcsp:: getOCSPResponse()	取得した CVS 応答から OCSP 群 LpkOcsps クラスを取得する。
Step.9	LePKI::addOcsp()	全取得 OCSP をループして順番に独自証明書ストアに追加する。
Step.10	LePAdES::setPki()	独自証明書ストアをセットした LePKI インタンスを LePAdES クラスにセットする。
Step.11	LePAdES::verify()	検証を実行する。この時に getCvs 取得時に指定したトラストアンカー証明書を rootCert 引数に指定する。 ※　rootCert 引数は V1.08.R1 からサポート。
Step.12	LePAdES.addLTV()	Step.11 の検証結果 XML を使って検証情報埋め込み。

```
////////////////////////////////////////////////////////////////////////
// ※ 本サンプルの実行には V1.08.R1 以降が必要です。
// ※ 本サンプルでは見やすくする為にエラーチェックを省いています。

////////////////////////////////////////////////////////////////////////
// CVS 処理
le::LpkUtil util;
le::LpkCert root;                           // CVS のルート証明書（必須）
le::BINARY bin;

// トラストアンカーの指定（GPKI のルート証明書）
rc = root.setFile(cvsRoot.c_str());
// CVS での検証実行
le::FLAG opt = 1;                           // CVS への CERTs/CRLs/OCSPs の取得指定
rc = util.getCvs(bin, cert, root, opt, cvsUrl.c_str());

////////////////////////////////////////////////////////////////////////
// CVS の解析
int stat;
le::LpkOcsp cvs;          // CVS は OCSP クラスにセットして解析する
le::LpkCerts certs;
le::LpkCrls crls;
le::LpkOcsps ocsps;

// CVS 結果のセット
rc = cvs.setBin(bin);
if (rc < 0)
// CVS のステータス確認
stat = cvs.getCertPathStatus();
if (stat != 0)
{
        std::cout << "get CVS status error(" << rc << ")." << std::endl;
        return Exit(1);
}
// CVS 情報から証明書の取得とセット
certs = cvs.getCertPath();
for (size_t i = 0; i < certs.size(); i++) {
        std::cout << "SET CVS CERT[" << i << "]" << std::endl;
        pki.addCert(certs[i]);
}
// CVS 情報から CRL の取得とセット
crls = cvs.getRevocationList();
for (size_t i = 0; i < crls.size(); i++) {
        std::cout << "SET CVS CRL[" << i << "]" << std::endl;
        pki.addCrl(crls[i]);
}
// CVS 情報から OCSP の取得とセット
ocsps = cvs.getOCSPResponse();
for (size_t i = 0; i < ocsps.size(); i++) {
        std::cout << "SET CVS OCSP[" << i << "]" << std::endl;
        pki.addOcsp(ocsps[i]);
}
```

```
////////////////////////////////////////////////////////////////////////
// 署名付与
le::PdaField field;
le::LeString fieldName = "SIGN1";          // 署名フィールド名

// 署名付与用のファイル入力
rc = pades.loadFile(inputFile.c_str(), NULL, NULL);

// 署名フィールド
field.setName(fieldName.c_str());                    // 必須
rc = pades.addField(field);

// 署名付与
rc = pades.addEnhanced(fieldName.c_str(), cert);
// リロード
rc = pades.saveFile((const char*)NULL, true, true, NULL);

////////////////////////////////////////////////////////////////////////
// 検証情報の埋め込み
le::BINARY xml;

// PKI のセット(CVS の結果をセット済み)
rc = pades.setPki(pki);

// 検証実行
le::FLAG vflag = le::PVF_NONE;
bool value = true;        // 検証情報のバイナリデータを出力(LTV に利用する場合には true にする)
rc = pades.verify(xml, value, NULL, NULL, &root, vflag);
// 検証情報埋め込み
rc = pades.addLTV(xml);

// LTV 化 PDF ファイル出力
rc = pades.saveFile(outputFile.c_str(), false, true, NULL);
```

C++による CVS 利用例（sample/LePAdES/cpp/CppCvs.cpp を参照）

５．設定情報ＸＭＬ

LE:PAdES:Lib では固定された設定ファイルは無い。ただし操作上良く使われる情報はクラス毎に異なる XML 形式にて入出力が可能になっている。LpaCmd の引数「-xml」を指定することで XML の出力が可能であり、同じく「-set xmlfile」により XML の読み込みが可能である。

設定 XML の出力（-xml オプション）
＞ LpaCmd -field -name SIGN2 -page 1 -rect 100 100 200 200 -xml <?xml version="1.0"?> <PdaField name="SIGN2">   <page>1</page>   <rect>100.000000 100.000000 200.000000 200.000000</rect>   <coordinate>LB</coordinate> </PdaField>
設定 XML の保存（-xml オプションを付け、標準出力をリダイレクト）
＞ LpaCmd -field -name SIGN2 -page 1 -rect 100 100 200 200 -xml > field.xml
設定 XML の指定（-set オプション）
＞ LpaCmd -field -set field.xml -in input.pdf -out output.pdf

署名フィールド生成における設定 XML の作成と利用例

５．１．署名フィールド XML 仕様（PdaField）

PdaField		ルート	署名フィールドの情報用の XML
	name	属性	署名フィールド名
	page	要素	ページ番号
	rect	要素	ページ内の位置(x1, y1, x2, y2)
	coordinate	要素	基点指定(LB:左下[標準]、LT:左上、RB:右下、RT:右上)

５．２．署名辞書 XML 仕様（PdaSign）

PdaSign		ルート	署名辞書用の XML
	type	属性	署名種類(pkcs7/atach/docts/cades)
	name	要素	署名フィールド名
	mdp	要素	署名種別(none/1/2/3) none： 普通署名 MDP 署名を使わない 1: 証明署名 変更を許可しない 2: 証明署名 フォームフィールド入力と署名フィールドに署名 3: 証明署名 注釈作成、フォームフィールド入力と署名フィールドに署名 4: ロック署名 変更を許可しない 5: ロック署名 フォームフィールド入力と署名フィールドに署名 6: ロック署名 注釈作成、フォームフィールド入力と署名フィールドに署名
	hash	要素	署名ハッシュ方式(sha1/s256/s384/s512)
	location	要素	署名場所
	reason	要素	署名理由
	contact	要素	コンタクト先
	commonName	要素	一般名
	now	要素	署名時刻
	opt	要素	オプション(none/nochn|prevf) none: 署名オプション無し nochn: 署名証明書のチェーンを埋め込まない (非推奨) prevf: 署名前に署名証明書の検証を行う (検証に成功時のみ署名可)
	filter	要素	自分で設定する/Filter 指定
	LpkTimestamp	要素	署名タイムスタンプ指定(5．6．参照)

５．３．署名外観 XML 仕様（PdaAppearance）

PdaAppearance		ルート	署名外観の情報用の XML
	grap	要素	グラフィック指定(text/name/image/pdf) text: 無し(テキスト外観のみ) name: 証明書の commonName 表示 image: 画像イメージ(サポート形式=JPEG/BMP/PNG) pdf: PDF イメージ(最初の画像/コンテンツを利用)
	text	要素	テキスト指定(none/name|date|reason|locat|subj|label|logo) none: テキスト外観無し or 指定: name=名前 | date=日付 | reason=理由 | locat=署名場所 | subj=識別名 | label=ラベル(説明) | logo=ロゴ
	image	要素	署名外観のイメージ(JPEG/BMP/PNG/PDF)ファイルの指定
	logo	要素	署名外観のグロゴ(背景)ファイルの指定(現在未サポート)
	font	要素	フォント指定(現在未サポート、"MS-Mincho"固定)
	format	要素	テキスト日付の日時フォーマット指定(例:"YYYY/MM/DD hh:mm:ss Z")

５．４．証明書 XML 仕様（LpkCert）

LpkCert		ルート	証明書指定用の XML
	type	属性	証明書指定方法(p12/finger) p12: PKCS#12 指定 ファイルとパスワードが必要 finger: Windows 証明書ストアから指紋(HEX 文字列)で指定 x509: Windows 証明書ストアの証明書をバイナリ形式で指定 card: IC カード利用 p11: PKCS#11 の IC カード ドライバファイルとパスワードが必要 ※Linux 版では “p12” のみ利用可能
	file	要素	証明書ファイルのパス(p12 または x509 の時に利用)
	passwd	要素	PKCS#12 ファイル用のパスワード(p12 の時に利用)
	enc	属性	パスワード暗号化の指定、未指定なら暗号化されない平文になる
	finger	要素	証明書のハッシュ値(指紋)で指定
	cardType	要素	PKI の IC カードの指定 （※Windows 版のみ利用可能） capi: CryptoAPI 利用(GPKI 等) 証明書選択画面から指定 jpki: JPKI 公的個人認証 IC カード（ハッシュ方式は現在 SHA1 のみ対応） hpki: HPKII ヘルスケア PKI IC カード（※実装済みだが動作は未検証） p11: PKCS#11 利用 （※現在未実装）

５．５．タイムスタンプ XML 仕様（LpkTimestamp）

LpkTimestamp		ルート	タイムスタンプ指定用の XML
	type	属性	タイムスタンプ種別(3161/amano) ※ amano は現在未サポート
	url	要素	タイムスタンプ局の URL
	hash	要素	タイムスタンプハッシュ方式(sha1/s512)
	id	要素	基本認証用ユーザ ID
	passwd	要素	基本認証用パスワード
	enc	属性	パスワード暗号化の指定、未指定なら暗号化されない平文になる

５．６．署名検証 XML 仕様（LePKI）

LePKI		ルート	署名検証の情報用の XML
	ver	属性	LePKI のバージョン番号(参考情報で利用されない)
	Store	要素	独自証明書ストア用ディレクトリ指定
	flag	属性	証明書ストア設定フラグ(none/org|win|nocache) none: 証明書ストア無し(非推奨) org: 独自証明書ストア利用 win: Windows 証明書ストア利用 nocache: CRL キャッシュを利用しない
	Cache	要素	CRL キャッシュ用ディレクトリ指定
	time	属性	CRL キャッシュ保持時間(秒)、標準 0 秒(オフ)
	Repository	要素	ディレクトリサーバ指定(現在未サポート)
	Validation	要素	証明書検証サーバ指定(現在未サポート)

５．７．署名 XML 仕様（LePAdES：複数の情報を一括して扱う）

LePAdES		ルート	署名用の XML(実はルートは別の名称でも構わない)
	ver	属性	LePAdES のバージョン番号(参考情報で利用されない)
	PdaField	要素	署名フィールド情報(5．1．参照)
	PdaSign	要素	署名辞書情報(5．2．参照)
	PdaAppearance	要素	署名外観情報(5．3．参照)
	LePKI	要素	署名検証情報(5．4．参照)
	LpkCert	要素	署名証明書情報(5．5．参照)
	LpkTimestamp	要素	ドキュメントタイムスタンプ指定(5．6．参照)
	contsize	要素	署名データサイズ(標準 10240 バイト)

６．検証情報ＸＭＬ

６．１．検証結果 XML 仕様（PdaVerifyXml）

[DUMMY]	ルート	検証結果によりルート要素名は異なる None：署名無し Verified：有効［VALID］ Indeterminate：不明［INDETERMINATE］（検証情報の不足） Failed：無効［INVALID］（失効や改ざん等） ERROR：その他エラー
Sign	要素	署名情報
hashtype	属性	※1 署名方式のハッシュアルゴリズム
signtype	属性	※13 署名方式のアルゴリズム（V1.09.B1 より）
hash	属性	署名データのハッシュ値（ゼロを含む）、VRI 指定に使う
name	属性	※2 署名フィールド名
subfilter	属性	署名データの種類を示す SubFilter 値 /adbe.pkcs7.detached：PAdES-Basic の PKCS#7 署名 /adbe.pkcs7.sha1：PAdES-Basic の PKCS#7 署名（ハッシュ） /ETSI.CAdES.detached：PAdES-Enhanced の CAdES 署名
byterange	属性	署名辞書の ByteRange 配列値
Verify	要素	署名検証結果
status	属性	※3 検証結果ステータス
date	属性	※4 署名タイムスタンプまたは署名日時の情報
Certs	要素	署名証明書の認証パス
status	属性	※3 検証結果ステータス
Cert	要素	証明書
name	属性	※5 証明書一般名
status	属性	※3 検証結果ステータス
valid	属性	※6 証明書検証種類（CRL か OCSP か）
from	属性	※7 情報取得場所
limit	属性	※8 証明書有効期限
id	属性	※9 証明書のシリアル番号（HEX 文字列）
Value	要素	証明書のバイナリ値
Crl	要素	CRL（証明書失効リスト）
name	属性	※10 CRL 発行者一般名
from	属性	※7 情報取得場所
update	属性	※11 発行日時
Value	要素	CRL のバイナリ値
Ocsp	要素	OCSP（オンライン証明書状態プロトコル）
name	属性	※12 署名証明書一般名
from	属性	※7 情報取得場所

					update	属性	※11 発行日時
					Value	要素	OCSP のバイナリ値
		SigTS				要素	署名タイムスタンプ（署名データに含まれるタイムスタンプ）
			hashtype			属性	※12 タイムスタンプのハッシュアルゴリズム
			Verify			要素	署名タイムスタンプ検証結果
				status		属性	※3 検証結果ステータス
				~~date~~		~~属性~~	~~署名タイムスタンプ日時~~ 注：V1.06.R2 以降は廃止となった Sign/Verify/date を利用。
			Certs			要素	署名タイムスタンプ署名証明書の認証パス
				status		属性	※3 検証結果ステータス
				Cert		要素	証明書
					name	属性	※5 証明書一般名
					status	属性	※3 検証結果ステータス
					valid	属性	※6 証明書検証種類（CRL か OCSP か）
					from	属性	※7 情報取得場所
					limit	属性	※8 証明書有効期限
					id	属性	※9 証明書のシリアル番号（HEX 文字列）
					Value	要素	証明書のバイナリ値
					Crl	要素	CRL（証明書失効リスト）
						name	属性	※10 CRL 発行者一般名
						from	属性	※7 情報取得場所
						update	属性	※11 発行日時
						Value	要素	CRL のバイナリ値
					Ocsp	要素	OCSP（オンライン証明書状態プロトコル）
						name	属性	※12 署名証明書一般名
						from	属性	※7 情報取得場所
						update	属性	※11 発行日時
						Value	要素	OCSP のバイナリ値
	DocTS					要素	ドキュメントタイムスタンプ情報
		hashtype				属性	※12
		hash				属性	タイムスタンプトークンのハッシュ値、VRI 指定に使う
		name				属性	※2 署名フィールド名
		byterange				属性	署名辞書の ByteRange 配列値
		Verify				要素	ドキュメントタイムスタンプ検証結果
			status			属性	※3 検証結果ステータス
			date			属性	ドキュメントタイムスタンプ日時
		Certs				要素	ドキュメントタイムスタンプ署名証明書の認証パス
			status			属性	※3 検証結果ステータス
			Cert			要素	証明書

				name		属性	※5 証明書一般名
				status		属性	※3 検証結果ステータス
				valid		属性	※6 証明書検証種類（CRL か OCSP か）
				from		属性	※7 情報取得場所
				limit		属性	※8 証明書有効期限
				id		属性	※9 証明書のシリアル番号（HEX 文字列）
				Value		要素	証明書のバイナリ値
				Crl		要素	CRL（証明書失効リスト）
					name	属性	※10 CRL 発行者一般名
					from	属性	※7 情報取得場所
					update	属性	※11 発行日時
					Value	要素	CRL のバイナリ値
				Ocsp		要素	OCSP（オンライン証明書状態プロトコル）
					name	属性	※12 署名証明書一般名
					from	属性	※7 情報取得場所
					update	属性	※11 発行日時
					Value	要素	OCSP のバイナリ値

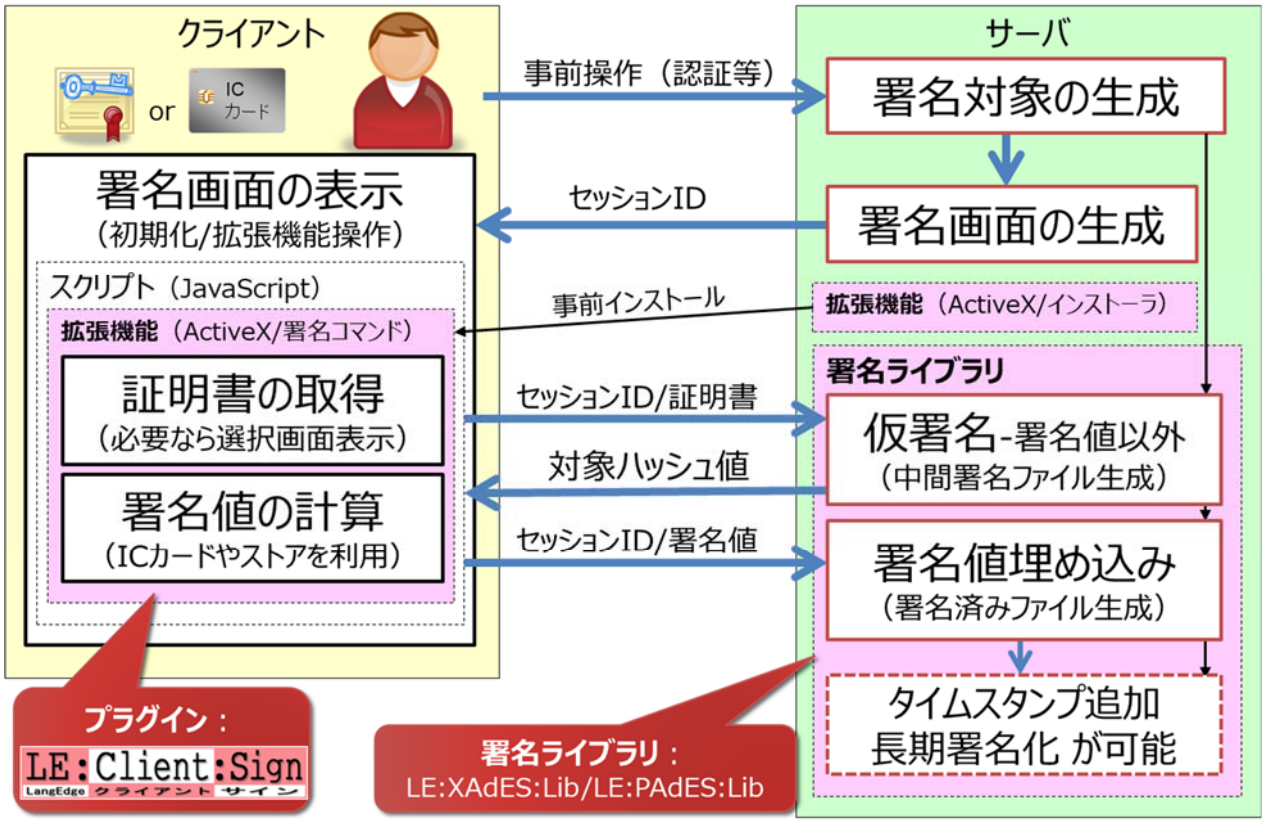
※1 署名方式のハッシュアルゴリズム	署名におけるハッシュアルゴリズム sha1、sha256、sha384、sha512
※2 署名フィールド名	署名データの署名フィールド名
※3 検証結果ステータス	Verified：有効 Indeterminate：不明（検証情報の不足） Failed：異常（失効や改ざん等） ERROR：その他エラー
※4 署名タイムスタンプまたは署名日時の情報	署名タイムスタンプがあればその日時 署名タイムスタンプが無ければ署名辞書の日時
※5 証明書一般名	証明書の CommonName
※6 証明書検証種類（CRL か OCSP か）	CRL：検証は CRL による OCSP：検証は OCSP による
※7 情報取得場所	unknown：不明 data：データ(署名データ等)自身から取得 orgStore：独自証明書ストアから取得 addStore：独自証明書ストアに後から追加した情報 winStore：Windows 証明書ストアから取得 network：ネット接続から取得(キャッシュも含む) valid：検証データ(OCSP か CRL 署名書)から取得 pdfDss：PDF の DSS 辞書から取得
※8 証明書有効期限	証明書の失効日時
※9 証明書のシリアル番号（HEX 文字列）	証明書のシリアル番号
※10 CRL 発行者一般名	CRL の Issuer 情報
※11 発行日時	CRL/OCSP の発行日時
※12 署名証明書一般名	OCSP の署名証明書の CommonName

※13 署名アルゴリズム名	V1.09.B1 より ECDSA 対応 rsa-s1、rsa-s256、rsa-s384、rsa-s512、 ecdsa-s1、ecdsa-s256、ecdsa-s384、ecdsa-s512

7．クライアント署名（Ver2.1）

現在サポートしているクライアント署名は Ver2.1 となり、別プロダクト LE:Client:Sign（LeClientSign）となる。この為マニュアルも「クライアント署名 V2.1 マニュアル」として別途提供される為にここでは概要のみ開設する。クライアント署名 V2.1 のマニュアルや実行環境は clientV2 ディレクトリ下に格納されている。クライアント署名 Ver2.0 以降では実行環境として、ActiveX を使った IE11（V1.0 サポート済み）と、署名コマンドを使った Edge/Chrome が利用可能となっている。

クライアント署名 V2 マニュアル　clientV2/doc/LeClientSign2-manual.pdf



クライアント署名概要

クライアント署名 V2 のライセンス：

ご契約ライブラリ	ご契約ライセンス	クライアント署名 V2 ライセンス
XML 長期署名 LE:XAdES:Lib	サーバ/開発ライセンス	ライセンス範囲（1 サーバ）内で利用可能。
	OEM ライセンス	LE:Client:Sign ライセンスが**別途必要**。※1
	ASP ライセンス	利用可能。

※1：ご利用される場合にはお問合せください。

７．１．ブラウザ（ActiveX プラグインと署名コマンド）の選択

クライアント署名 V2 には、ActiveX プラグイン（コンポーネント）と署名コマンド（カスタム URL スキーム）の 2 種類があり、それぞれ利用可能となるブラウザが異なる。利用者にどちらを利用させるかを決める必要がある。

ActiveX プラグインは IE11 のみで動作する。署名コマンドは OS 機能のカスタム URL スキームを利用して実行する為にほぼ全てのブラウザで利用することが可能だがブラウザ依存もあるので現在では Chrome と Chromium ベースの Edge（新 Edge）のみ動作保証をしている。

どちらも JavaScript から実行するが、署名コマンドの利用には JavaScript ライブラリ（LeCSignCmdWS.js）の API を提供する。

	Active X プラグイン	署名コマンド（カスタム URL）
クライアント OS 対応	Windows 10（Windows 8.1）	
ブラウザ対応	Internet Explorer 11 （IE11）	Chrome / 新 Edge ※ レガシ Edge は対象外
事前インストール	不要 ※ 実行時自動インストール	必要（インストーラ提供） ※ exe ファイル配置とレジストリ登録
提供ファイル	LeCSignCapiA.cab / LeCSignP11A.cab	LeCSignCmd.exe / LeCSignCmdWS.js （ LeCSignCmdSetup.msi ）
サーバ連携	必要（サーバ側に LE:PAdES:Lib か LE:XAdES:Lib が必要） ※ 署名部のサーバ側実装は V1 のままで共通利用が可能	
呼び出し方法	object 要素の API 呼び出し	実行用 JavaScript API 提供 ※ 実行後に WebSocket 通信
署名結果確認方法	API の戻り値で確認可能	
ライセンス	LE:Client:Sign V2 ライセンスが必要 ※ Chrome/Edge 対応は標準機能の為に別料金は不要	

ラング・エッジ LE:Client:Sign V2 の機能

７．２．CAPI（CryptoAPI）と PKCS#11 の選択

Windows 環境の署名用 API として、PKCS#11 と CAPI（CryptoAPI）の 2 種類が良く使われる。どちらを使うかは利用する秘密鍵の利用形態に依存する。例えば IC カードの場合には PKCS#11 の API のみ利用可能な場合がある。つまりどちらを利用するかは、利用する証明書と秘密鍵の提供形態に依存する。ActiveX プラグインではコンポーネント自体が異なり、署名コマンドでは初期化フラグでどちらを利用するかを指定する。

	CAPI（CryptoAPI）	PKCS#11
Windows	Windows 標準の証明書ストアと IC カード	DLL ファイル経由で利用
初期化情報	通常は特に情報は必要ないが、CSP 名の指定が必要となる IC カードの場合には CSP 名他の情報を与える	DLL ファイルのパスを与える
IC カード	CAPI 対応の IC カード	PKCS#11 対応の IC カード
補足	PKCS#12 形式で秘密鍵と証明書が提供されている場合には、Windows 証明書ストアにインストールして利用可能	HPKI 等では PKCS#11 の利用が推奨されている

CAPI と PKCS#11 の違い

ActiveX プラグインは CAPI と PKCS#11 それぞれ異なるコンポーネントを利用する。署名コマンド初期化時の引数により CAPI と PKCS#11 を切り替えて利用する。

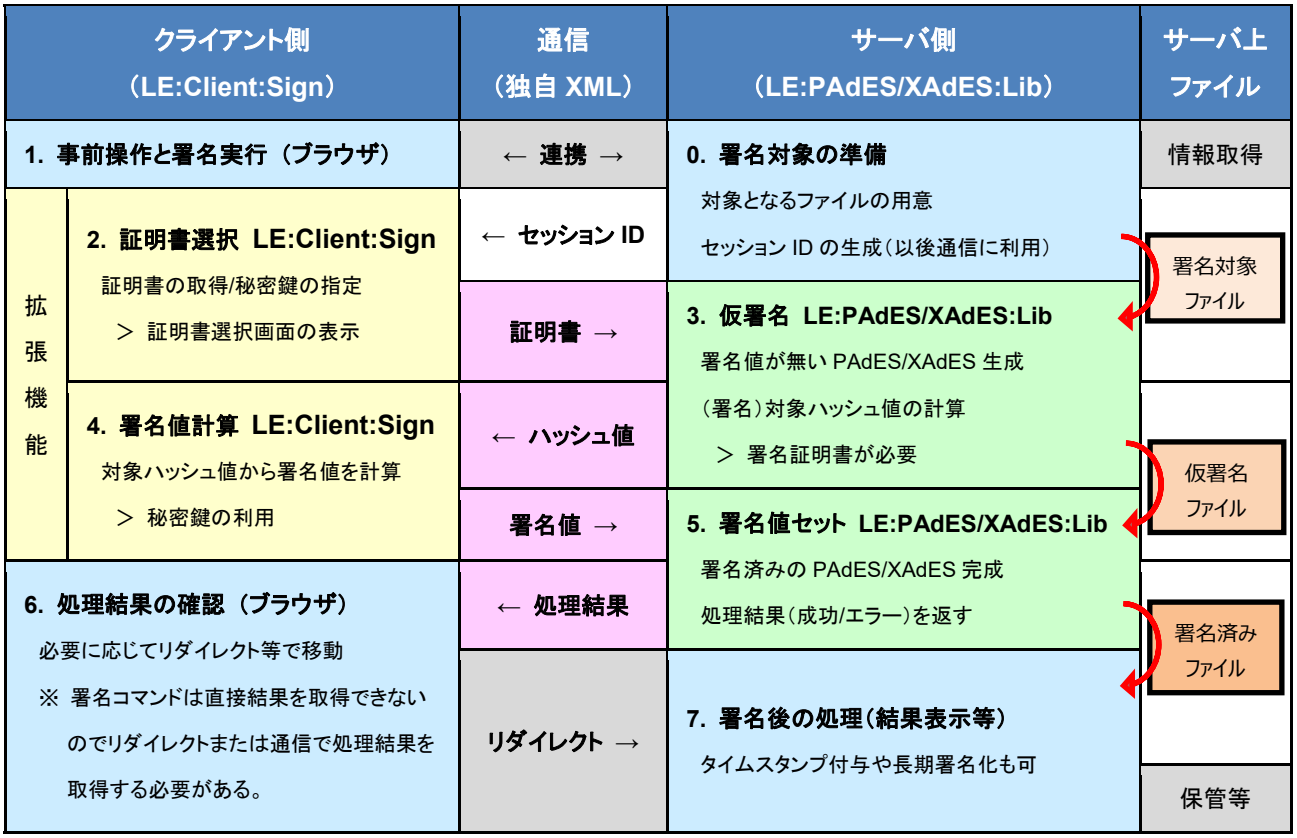
	CAPI（CryptoAPI）	PKCS#11
ActiveX プラグイン	LeCSignCapiA.cab	LeCSignP11A.cab
署名コマンド	LeCSignCmd.js（LeCSignCmdSetup.msi）	

ActiveX プラグインと署名コマンドの提供モジュール

７．３．署名サーバー連携

クライアント署名は署名サーバーと連携して署名処理を行う。サーバーとクライアント間の通信プロトコルは XML を使った独自仕様となっている。サーバー側にて独自 XML 通信プロトコルに対応した実装を行う必要がある。またサーバー側では取得した通信結果から「仮署名」と「署名値セット」の処理を実装する必要がある。

またサーバー側のライブラリを PDF 署名用の LE:PAdES:Lib（LE:PAdES-Basic:Lib も可）を利用するのか、XML 署名用の LE:XAdES:Lib を利用するのかを決めておく必要がある。なお LE:PAdES:Lib では独自通信プロトコルに対応したクラス PdaClientXml を、LE:XAdES:Lib では XdaClientXml を提供している。



クライアントとサーバー間の連携とサーバー側の処理

※ LE:Client:Sign Ver2.1 よりサーバー連携せずクライアント側で XAdES-BES の生成（ローカル署名）が可能となった。XAdES-BES の生成には API として sign() では無く xades() を利用する。Xades を使う場合も cert() の事前呼び出しは必要となる。

７．４．署名サーバー実装例

１）実装サンプル：Tomcat 用の Java 実装例

clientV2/bin_cmd/server_pades/PLeCSignCert.java ：証明書取得時
clientV2/bin_cmd/server_pades/PLeCSignSign.java ：署名値取得時
clientV2/bin_cmd/server_pades/PLeCSignShow.java ：結果表示時

証明書取得時の仮署名の C++実装例：

```
// ------------------------------------------------------------------------
// 証明書取得対応処理
std::string cert(le::PdaClientXml& inXml, std::string& dir)
{
	int rc = 0;
	std::string outXml, inputPdf, fieldName, outputPdf;
	le::LpkCert cert;
	le::PdaField field;
	le::LePAdES pades;
	le::BINARY certBin, hash, outBin;

	// 試験設定
	fieldName = "LCS1";
	inputPdf = dir + "input.pdf";
	outputPdf = dir + inXml.getSessionId().c_str() + "-temp.pdf";

	// 署名証明書の準備
	certBin = inXml.getData();
	rc = cert.setBin(certBin, false);
	if(rc < 0)
		return outXml;

	// 元 PDF ファイルの読み込み
	rc = pades.loadFile(inputPdf.c_str(), NULL, NULL);
	if(rc < 0)
		return outXml;

	// 署名フィールドの指定
	field.setName(fieldName.c_str());
	rc = pades.addField(field);
	if(rc < 0)
		return outXml;

	// V2 クライアント署名用の仮署名
	rc = pades.makeEnhanced2(fieldName.c_str(), cert, hash);
	if(rc < 0)
		return outXml;

	// 仮署名 PDF ファイルの保存
	rc = pades.saveFile(outputPdf.c_str());
```

```
        if(rc < 0)
                return outXml;

        // 応答 XML の生成
        outBin = inXml.createCertResponse(hash);
        if(outBin.empty())
                return outXml;
        outXml.assign((const char*)&outBin[0], outBin.size());

        return outXml;
}
```

2）LpaCmd のダミークライアント -client 引数による実装例

// Windows

sample/LaPAdES/cmd/CmdClientTest.bat : シンプルな利用方法

sample/LaPAdES/cmd/CmdClientTest2.bat : 署名外観/TS 付き利用方法

sample/LaPAdES/java/JavaClientTest.bat : Java 試験

// Linux

sample/LaPAdES/cmd/CmdClientTest.sh : シンプルな利用方法

sample/LaPAdES/cmd/CmdClientTest2.sh : 署名外観/TS 付き利用方法

sample/LaPAdES/java/JavaClientTest.sh : Java 試験

// Java

sample/LaPAdES/java/LePAdES_client.java : Java からの実装例

```
// クライアント側 (LCS_CERT) : 証明書選択 > 証明書 XML
$ LpaCmd.sh -client -cert p12 LeTest.p12 test -sid T01 > CERT.xml
// サーバー側（前準備）: 署名フィールド生成
$ LpaCmd.sh -field -in input.pdf -out field.pdf
// サーバー側 (LCS_HASH) : 仮署名 < 証明書 XML > ハッシュ値 XML
$ LpaCmd.sh -server -in field.pdf -out make.pdf < CERT.xml > HASH.xml
// クライアント側 (LCS_SIGN) : 署名値計算択 < ハッシュ値 XML > 署名値 XML
$ LpaCmd.sh -client -cert p12 LeTest.p12 test -sid T01 < HASH.xml > SIGN.xml
// サーバー側 (LCS_RSLT) : 署名値埋め込み < 署名値 XML > 結果 XML
$ LpaCmd.sh -server -in make.pdf -out sign.pdf < SIGN.xml > RSLT.xml
// 結果確認（本来はクライアント側で確認）
$ cat RSLT.xml
```

LpaCmd を使ったクライアント署名の利用例（クライアント側はダミー実行）

７．５．LE:Client:Sign フォルダ構成

LE:PAdES:Lib のリリースには LE:Client:Sign のバイナリ版が clientV2 として同梱される。バイナリ版であるのでソースコードは提供されない。また LE:Client:Sign のライセンスは別途となるので注意が必要となる。ソースコードは LE:Client:Sign 製品版にて取得可能。

フォルダ/ファイル	説明	提供
readme-LeClientSign2.txt	LE:Client:Sign V2 リリースノート	バイナリ版
[bin_activex]	ActiveX モジュール及び試験ファイル	バイナリ版
LeCSignCapiA.cab	CAPI 用 ActiveX プラグイン	バイナリ版
LeCSignP11A.cab	PKCS#11 用 ActiveX プラグイン	バイナリ版
LeCSignIE11.js	LE:Client:Sign V2 フラグ定義 JavaScreipt	バイナリ版
*.htm	IE11 用試験ファイル	バイナリ版
unregistcapi.bat	CAPI 用 ActiveX プラグイン登録解除バッチ	バイナリ版
[bin_cmd]	署名コマンドモジュール及び試験ファイル	バイナリ版
LeCSignCmd.exe	署名コマンド実行ファイル（試験用）	バイナリ版
LeCSignCmdSetup.zip LeCSignCmdSetup.msi	署名コマンドインストーラ（実行用）	バイナリ版
LeCSignCmdWS.js	署名コマンド API 定義 JavaScript	バイナリ版
*.html	Edge/Chrome 用試験ファイル	バイナリ版
Regist.bat Unregist.bat	bin_cmd/LeCSignCmd.exe の登録と解除用のバッチファイル（非インストール試験用）	バイナリ版
[server_pades]	LE:PAdES:Lib のサーバ側実装 Java サンプル	バイナリ版
[ServerTest]	ローカル試験用のダミーサーバ用フォルダ	バイナリ版
[bin]	ダミーサーバ実行環境 ※PATH 環境変数に追加	バイナリ版
[doc]	マニュアル等のドキュメント	バイナリ版
LeClientSign2-manual.pdf	マニュアル（本ファイル）	バイナリ版
*.pdf	補足ドキュメント	バイナリ版

８．ＰＤＦ操作

本章では PDF 電子署名とは直接関係が無いが、PDF の操作を行う機能について説明をする。Ver1.04.R1 より PDF 操作のみを行う為にコマンドの主操作指定に -pdf 引数が追加された。

８．１．作成日（CreationDate）と更新日（ModDate）の指定

PDF の文書情報辞書には、作成日（CreationDate キー）と更新日（ModDate キー）を付けることができる。通常作成日は変更できないが、別途 PDF ファイルを作成した場合に署名時刻を作成日にするような運用も考えられる。この為に Ver1.04.R1 より作成日と更新日の指定が可能となった。なお更新日に関しては、Ver1.04.R1 以前より署名時にはコンピュータの現在時刻でセットしていた。指定はコマンドラインの -credate と -moddate 引数か、C++/Java の LePAdES::setDates() を利用して行う。作成日と更新日の片方だけの指定も可能であるが、指定時には組み合せにより以下のような仕様になる。

説明	作成日	更新日	保存される PDF の日付
指定しない場合 （デフォルト）	× 未指定	× 未指定	作成日：元ファイルから変更されない 更新日：保存時の PC 時刻に変更される
作成日のみ指定	○ 指定	× 未指定	作成日：指定作成日がセットされる 更新日：指定作成日と同じ時刻がセットされる
更新日のみ指定	× 未指定	○ 指定	作成日：元ファイルから変更されない 更新日：指定更新日がセットされる
作成日と更新日 の両方を指定	○ 指定	○ 指定	作成日：指定作成日がセットされる 更新日：指定更新日がセットされる

作成日と更新日の指定

日付は文字列として与えるが、日付文字列の仕様は PDF の日付仕様（ISO/IEC 8824 / ASN.1）に準拠している。D:YYYYMMDDHHmmSSOHH'mm' として指定する。各意味は以下の通り。

D: ：日付の接頭辞（本来オプションだが LE:PAdES:Lib では必須）
YYYY：年（西暦で 4 桁必須） MM ：月（01 - 12） DD ：日（01 - 31）
HH ：時間（00 - 23） mm ：分（00 - 59） SS ：秒（00 - 59）
O ：世界時との関係（+ は UT より遅い、– は UT より早い、z は UT 時刻を示す）
HH’mm’：O が+か–の場合に差分を、HH は時間（00-23）と mm は分（00-59）で指定

例えば日本時間（JST、UTC+9 時間）の 2015 年 1 月 10 日午後 13 時 28 分 45 秒なら以下となる。HH と mm にはカンマ ‘ が必須なので忘れないようにする。

D:20150110132845+09’00’

作成日や更新日の指定をコマンドから行う場合のサンプルを以下に示す。

○ 署名付与時に作成日を指定（更新日も同じ日時になる）

```
> LpaCmd -sign -newf -cert p12 LeTest.p12 test -credate D:20150110132845+09'00' ¥
  -in input.pdf -out credate.pdf
```

○ 署名付与時に更新日を指定（作成日は変更されない）

```
> LpaCmd -sign -newf -cert p12 LeTest.p12 test -moddate D:20150110132845+09'00' ¥
  -in input.pdf -out moddate.pdf
```

○ 署名付与時に作成日と更新日を指定

```
> LpaCmd -sign -newf -cert p12 LeTest.p12 test -credate D:20150110132845+09'00' ¥
  -moddate D:20150120000000+09'00' -in input.pdf -out moddate.pdf
```

以下に C++と Java の LePAdES クラスの API を示す。

```
/** 保存時にセットする作成日や更新日を指定する(オプション)
 *
 * @param creationDate [IN] 作成日の文字列を指定(NULL なら指定しない)
 * @param modDate [IN] 更新日の文字列を指定(NULL なら指定しない)
 * @retval マイナス値 エラーコードが返る
 * @note creaionDate を指定して modDate が未指定の場合は modDate は creationDate になる
 * @note 日付書式 "D:YYYYMMDDHHmmSSOHH'mm'" 例) "D:20150107122030+09'00'"
 */
 int setDates(const char* creationDate, const char* modDate);
```

C++の作成日と更新日のセット API

```
/** 保存時にセットする作成日や更新日を指定する(オプション)
 *
 * @param creationDate [IN] 作成日の文字列を指定(null なら指定しない)
 * @param modDate [IN] 更新日の文字列を指定(null なら指定しない)
 * @retval マイナス値 エラーコードが返る
 * @note creaionDate を指定して modDate が未指定の場合は modDate は creationDate になる
 * @note 日付書式 "D:YYYYMMDDHHmmSSOHH'mm'" 例) "D:20150107122030+09'00'"
 */
 public native int setDates(String creationDate, String modDate);
```

Java の作成日と更新日のセット API

８．２．XMP（メタデータ）操作

PDF では XML 形式のメタデータ仕様である XMP を利用できる。XMP には独自の属性や情報を埋め込むことが出来る。独自の XMP を署名時に埋め込むような運用も考えられる。この為に Ver1.04.R1 より XMP の埋め込み指定が可能となった。また PDF に埋め込まれている XMP を取り出す事もできるうようになった。指定はコマンドラインの -setxmp と -getxmp 引数か、C++/Java の API の LePAdES::setXmp() と LePAdES::getXmp() を利用して行う。

なお埋め込み時に指定する XMP 情報はそのまま何の操作もせずバイナリ情報として埋め込むだけであるので、XML 正当性を含めたその内容は運用者が責任を持つ必要があるので注意すること。

XMP 埋め込みや取得の指定をコマンドから行う場合のサンプルを以下に示す。

○ 署名付与時に XMP ファイルを埋め込む

```
＞ LpaCmd -sign -newf -cert p12 LeTest.p12 test -setxmp new.xmp ¥
   -in input.pdf -out signxmp.pdf
```

○ PDF に XMP ファイルを埋め込む

```
＞ LpaCmd -pdf -setxmp new.xmp -in input.pdf -out newxmp.pdf
```

○ PDF から XMP ファイルを取得する

```
＞ LpaCmd -pdf -getxmp save.xmp -in input.pdf
```

以下に C++と Java と.NET の LePAdES クラスの API を示す。

```
/** XMP をメモリからセット(オプション)
 *
 * @param xml [IN] XMP バイナリ情報のポインタを指定
 * @param len [IN] XMP バイナリ情報のサイズを指定
 * @retval マイナス値 エラーコードが返る
 */
 int    setXmp(const BYTE* xmp, SIZE len);

/** XMP データを取得する(オプション)
 *
 * @param data [OUT] XMP データを取得する
 * @retval マイナス値 エラーコードが返る
 */
 int    getXmp(BINARY& data);
```

C++の XMP の埋め込みと取得の API

```
/** XMP をメモリからセット(オプション)
 *
 * @param xml [IN] XMP バイナリ情報を指定する
 * @retval マイナス値 エラーコードが返る
 */
 public native int      setXmp(byte[] xmp);

/** XMP データを取得する(オプション)
 *
 * @retval バイナリ配列 XMP データが返る
 */
 public native byte[]   getXmp();
```

Java の XMP の埋め込みと取得の API

```
/** XMP をメモリからセット(オプション)
 *
 * @param xml [IN] XMP バイナリ情報を指定
 * @retval マイナス値 エラーコードが返る
 */
int     setXmp(array<Byte>^ xmp);

/** XMP データを取得する(オプション)¥n¥n
 *
 * @retval バイナリ XMP データが返る
 */
array<Byte>^    getXmp();
```

.NET の XMP の埋め込みと取得の API

Ver1.05.R1 よりプロパティ操作の機能が追加された。プロパティ操作は XMP にも登録される。

C++/Java/.NET の API に以下を追加（LePAdES クラス）

プロパティ情報取得：getPropNum()/getPropName()/getPropValue()
プロパティ情報編集：setProp()/delProp()

LpdCmd には以下の引数を追加（-pdf のサブコマンド）

プロパティ情報取得：-prop [key]
プロパティ情報編集：-padd key¥value / -pdel key

８．３．添付ファイル（v1.06.R2 以降）

PDF には任意形式のファイルを添付ファイルとして PDF に埋め込む仕様がある。埋め込まれるファイルの管理は PDF 構造としては階層構造となっているが、LE:PAdES:Lib の機能としては 2 階層までのサポートとなっている。この為に 3 階層以上を利用した添付ファイルには未対応であるので汎用的な対応ではないので注意すること。なお Adobe Acrobat で作成した場合に本執筆時点では 2 階層以内であるので対応している。

LE:PAdES:Lib の添付ファイル機能は基本的には添付ファイルが無い PDF に添付ファイルを追加するか、Adobe Acrobat で作成された添付ファイル付き PDF へ添付ファイルを追加することに対応をしている。添付ファイルの保存に関しても同じとなる。もしうまく動作しない PDF があった場合には問い合わせください。

C++/Java/.NET の API に以下を追加（LePAdES クラス）

```
// 初期化
添付ファイル解析指示：setParseEmbed()　// PDF 読み込み時に添付ファイルの解析を行う ※
※ API 利用時は最初に setParseEmbed() を指定して PDF 読み込みする必要がある。
// 添付ファイル情報取得
添付ファイル数取得：getEmbedFileNum()
添付ファイル名取得：getEmbedFileName()
// 添付ファイル保存（外部ファイル出力）
添付ファイル保存：saveEmbedFile() / saveEmbedBinary()
// 添付ファイル追加（複数回呼び出しで複数ファイル添付が可能）
添付ファイル追加：addEmbedFile() / addEmbedBinary()
```

LpdCmd には以下の引数を追加（-pdf のサブコマンド）

```
添付ファイルのリスト表示：　-embedlist
添付ファイルを保存：　　-embedsave num [path]
　　path 省略時はそのままのファイル名で保存
　　num が-1 なら全添付ファイルをそのままのファイル名で保存
添付ファイルを追加：　　-embedadd path
```

※ 利用サンプル：sample/LePAdES/cmd/CmdPdfTest.bat（CmdPdfTest.sh）

８．４．PDF 情報（v1.06.R2 以降）

PDF 情報の詳細は「９．PDF 解析結果 XML」を利用します。ここでは簡易にページ数や PDF バージョンを取得する方法を説明する。

C++/Java/.NET の API に以下を追加（LePAdES クラス）

PDF ページ数取得：getPageNum()
PDF バージョン取得：getVersion()

LpdCmd には以下の引数を追加（-pdf のサブコマンド）

プロパティ情報取得：-pagenum
プロパティ情報編集：-version

９．ＰＤＦ解析結果ＸＭＬ（V1.06.R1 以降）

本章では Ver1.06.R1 より追加された PDF ファイルの解析機能について説明をする。

９．１．PDF 解析機能

PDF 解析機能は電子署名の検証では無いので外部接続はしないで PDF の構造のみ解析する。解析対象は以下の 3 つであり、標準では PDF 基本情報と PDF 署名情報が解析される。

１）PDF 基本情報 ： タイトルや作成者等の PDF プロパティ情報。
２）PDF 署名情報 ： 署名/タイムスタンプに関連する PKI/PAdES 情報。
３）PDF 画像情報 ： 各ページに含まれる画像情報(オプション)。

まず PDF 解析は LePAdES::parse() により行い、PDF 解析結果 XML 情報を取得する。取得した PDF 解析結果 XML は PdaParseXml クラスを使ってレポート文字列として取得することができる。解析フラグによりオプション指定が可能となっている。標準では PDF 画像情報と署名オプション情報はオフになっている。署名オプション情報は PPF_NO_SIGN_INFO が設定されていると取得されない。また PPF_ADD_VALUE フラグを使うことで証明書等を取り出すことも可能となっている。

```
/** 解析フラグ. */
typedef enum {
  PPF_NONE,           // フラグ指定無し（署名情報のみ返す、検証はしないで構造の情報のみ）
  PPF_NO_SIGN_INFO,   // PDF の署名情報は解析しない
  PPF_IMAGE_INFO,     // PDF のページ毎の画像情報を返す（主に電子帳簿保存法対応用が目的）
  PPF_SIGN_OPT_INFO,  // 署名オプション情報を返す（V1.08.R3 よりサポート）
  PPF_ADD_VALUE,      // 証明書等のデータや画像ストリームを出力（画像は HEX 化される）
} PDA_PARSE_FLAG;
```

```
/** PDF 情報を解析して XML 形式の PDF 解析結果 XML を返す
 * @param xml [OUT] PDF 解析 XML が返される
 * @param flag [IN] 解析フラグ(PDA_PARSE_FLAG)を指定(PPF_NONE:がデフォルト)
 * @retval マイナス値 エラーコードが返る
 * @note 出力された XML は PdaParseXml クラスで読み込んで情報取得が可能
 */
int LePAdES::parse(BINARY& xml, FLAG flag=PPF_NONE);
```

LpaCmd には -pdf 用に以下の引数を追加。

```
-parse [filepath] : PDF 解析結果 XML ファイルの出力(検証は行われない),
-pflag <nosign/image/value> : PDF 解析フラグ
   nosign : PDF の署名情報は解析しない
   image  : PDF のページ毎の画像情報を解析する
   value  : 証明書等の PKI データや画像のバイナリを出力する
   sigopt : 署名オプション情報を出力する
```

PDF 解析結果 XML レポート用クラス LpaParseXml の setXml() により PDF 解析結果 XML をセットし、getReport() により解析結果をレポート文字列として取得できる。

９．２．PDF 解析結果 XML 仕様（PdaParseXml）

LePdf							ルート	PDF 解析結果用の XML
	pFlag						属性	解析時のオプションフラグ
	fileName						属性	解析 PDF ファイル名
	Info						要素	**PDF 基本情報**
		title					属性	タイトル
		author					属性	作成者
		subject					属性	サブタイトル
		keywords					属性	キーワード
		creator					属性	アプリケーション
		producer					属性	PDF 変換
		creationDate					属性	作成日
		modDate					属性	更新日
	Signs						要素	**PDF 署名情報**
		DSS					要素	DSS 検証情報(PAdES 長期署名仕様)
			CERTs				要素	証明書群
				CERT			要素	証明書情報
					oid		属性	PDF オブジェクト ID
					name		属性	証明書一般名 (CommonName)
					issuer		属性	発行者名
					id		属性	証明書シリアル番号 (HEX 文字列)
					limit		属性	有効期限
					from		属性	情報取得場所 ※1
					crlDp		属性	CRL 発行場所(URL)
					ocspUrl		属性	OCSP 発行 URL
					Value		要素	証明書のバイナリ値 (HEX 文字列)
			CRLs				要素	CRL 群
				CRL			要素	CRL 情報
					oid		属性	PDF オブジェクト ID
					issuer		属性	発行者名
					thisUpdate		属性	発行(更新)日時
					nextUpdate		属性	次回更新日時(予定)
					from		属性	情報取得場所 ※1
					Value		要素	CRL のバイナリ値 (HEX 文字列)

			OCSPs					要素	OCSP（オンライン証明書状態プロトコル）群
				OCSP				要素	OCSP 情報
					oid			属性	PDF オブジェクト ID
					respId			属性	レスポンダ ID（HEX 文字列）
					atTime			属性	発行日時
					status			属性	ステータス ※2
					from			属性	情報取得場所 ※1
					certName			属性	証明書名ハッシュ値（HEX 文字列）
					certKey			属性	証明書鍵ハッシュ値（HEX 文字列）
					certId			属性	証明書シリアル番号（HEX 文字列）
					signerName			属性	OCSP 発行者名
					signerId			属性	OCSP 発行者シリアル番号（HEX 文字列）
					Value			要素	OCSP のバイナリ値（HEX 文字列）
			VRIs					要素	VRI 群(署名数ある)
				VRI				要素	VRI 署名情報(PAdES 長期署名仕様)
					hash			属性	署名データハッシュ値（HEX 文字列）
					CERTs			要素	VRI 証明書群
						CERT		要素	VRI 証明書情報
							oid	属性	VRI 証明書の PDF オブジェクト ID
					CRLs			要素	VRI の CRL 群
						CRL		要素	VRI の CRL
							oid	属性	CRL の PDF オブジェクト ID
					OCSPs			要素	VRI の OCSP 群
						OCSP		要素	VRI の OCSP 群
							oid	属性	OCSP の PDF オブジェクト ID
		Sign						要素	署名データと署名辞書の情報
			hash					属性	署名データハッシュ値（HEX 文字列）
			subFilter					属性	署名データの種類を示す SubFilter 値 /adbe.pkcs7.detached：PAdES-Basic の PKCS#7 署名 /adbe.pkcs7.sha1：PAdES-Basic の PKCS#7 ハッシュ署名 /ETSI.CAdES.detached：PAdES-Enhanced の CAdES 署名
			signTime					属性	署名タイムスタンプまたは署名辞書の署名日時の情報
			byteRange					属性	署名対象のバイトレンジ
			fieldName					属性	署名フィールド名
			page					属性	署名位置のページ番号
			rect					属性	署名位置の座標矩形
			CERTs					要素	署名データ中の証明書群
				CERT				要素	署名データ中の証明書情報 ▼1
					name			属性	証明書一般名（CommonName）

					issuer	属性	発行者名
					id	属性	証明書シリアル番号（HEX 文字列）
					limit	属性	有効期限
					from	属性	情報取得場所 ※1
					Value	要素	証明書のバイナリ値（HEX 文字列）
			CRLs			要素	署名データ中の CRL 群
				CRL		要素	署名データ中の CRL 情報
					issuer	属性	発行者名
					thisUpdate	属性	発行(更新)日時
					nextUpdate	属性	次回更新日時(予定)
					from	属性	情報取得場所 ※1
					Value	要素	CRL のバイナリ値（HEX 文字列）
			OCSPs			要素	署名データ中の OCSP 群
				OCSP		要素	署名データ中の OCSP 情報
					respId	属性	レスポンダ ID（HEX 文字列）
					atTime	属性	発行日時
					status	属性	ステータス ※2
					from	属性	情報取得場所 ※1
					certName	属性	証明書名ハッシュ値（HEX 文字列）
					certKey	属性	証明書鍵ハッシュ値（HEX 文字列）
					certId	属性	証明書シリアル番号（HEX 文字列）
					signerName	属性	OCSP 発行者名
					signerId	属性	OCSP 発行者シリアル番号（HEX 文字列）
					Value	要素	OCSP のバイナリ値（HEX 文字列）
					from	属性	情報取得場所 ※1
					Value	要素	CRL のバイナリ値（HEX 文字列）
			SigTS			要素	署名タイムスタンプ情報
				CERTs		要素	署名タイムスタンプの証明書群
					CERT	要素	署名タイムスタンプ中の証明書情報 ▼1 と同じなので属性省略
			SigOpt			要素	署名オプション情報（要 PPF_SIGN_OPT_INFO 指定）※3
				time		属性	署名日付：署名辞書の M キー値（システム時刻）※3
				name		属性	署名者名：署名辞書の Name キー値 ※3
				reason		属性	署名理由：署名辞書の Reason キー値 ※3
				location		属性	署名場所：署名辞書の Location キー値 ※3
				contact		属性	問合せ情報：署名辞書の ContactInfo キー値 ※3
		DocTS				要素	ドキュメントタイムスタンプ情報
			hash			属性	ドキュメントタイムスタンプ ハッシュ値（HEX 文字列）
			subFilter			属性	署名データの種類を示す SubFilter 値

					/ETSI.RFC3161：タイムスタンプトークン
			signTime	属性	ドキュメントタイムスタンプ日時の情報
			byteRange	属性	ドキュメントタイムスタンプ対象のバイトレンジ
			fieldName	属性	署名フィールド名
			page	属性	署名位置のページ番号
			rect	属性	署名位置の座標矩形
			CERTs	要素	ドキュメントタイムスタンプ中の証明書群
			CERT	要素	ドキュメントタイムスタンプ中の証明書情報 ▼1 と同じなので属性省略
	Pages			要素	**PDF 画像情報**（解析時オプション PPF_IMAGE_INFO が必要）
		Page		要素	ページ情報（1 ページ分の画像情報群）
			num	属性	ページ番号
			xSize	属性	ページ X 軸方向サイズ（単位:point）
			ySize	属性	ページ Y 軸方向サイズ（単位:point）
			rotate	属性	ページ回転（角度:0,90,180,270）
			Image	要素	画像情報（1 画像の情報）
				num	属性	画像番号（ページ内の連番）
				width	属性	画像幅（単位:pixel）
				height	属性	画像高さ（単位:pixel）
				colorsNum	属性	画像色数（1=GRAY/3=RGB/4=CMYK）
				colorsDepth	属性	画像色深度（256=8bit）
				xSize	属性	画像 X 軸方向サイズ（単位:point） PDF 情報から取得
				ySize	属性	画像 Y 軸方向サイズ（単位:point） PDF 情報から取得
				xRez	属性	画像 X 軸方向解像度（単位:dpi） JPEG から取得または計算値
				yRez	属性	画像 Y 軸方向解像度（単位:dpi） JPEG から取得または計算値
				filter	属性	画像フィルター名 /DCTDecode:JPEG 画像、/FlateDecode:PNG 画像
				name	属性	PDF 内部画像名（識別子）
				Value	要素	画像のバイナリ値 （HEX 文字列）

※1 情報取得場所	unknown：不明 data：データ(署名データ等)自身から取得 pdfDss：PDF の DSS 辞書から取得

※2 OCSP ステータス	OCSPResponseStatus ::= ENUMERATED {
	successful(0), --Response has valid confirmations（成功）
	malformedRequest(1), --Illegal confirmation request
	internalError(2), --Internal error in issuer
	tryLater (3), --Try again later
	--(4) is not used
	sigRequired (5), --Must sign the request
	unauthorized (6)--Request unauthorized
	}

※3 署名オプション情報	PPF_NO_SIGN_INFO が指定されていると出力されない。 V1.08.R2

付録Ａ．エラーコード

Ａ．１．LeUtil エラーコード（src/LeCommon/LeUtil.h）-1000 ～ -1099

LeUtil のエラーコードは全体を通して利用される汎用ユーティリティが返すエラーコードです。ただし表面に表示されることは無いはずです。

```
LPU_ERR_FILEWOPEN       = -1000,        // 書き込みファイルオープンエラー
LPU_ERR_FILEROPEN       = -1001,        // 読み込みファイルオープンエラー
```

Ａ．２．LpaCmd エラーコード（src/LePAdES/LpaCmd/LpaCmd.h）-1200 ～ -1399

LpaCmd のエラーコードはコマンドライン LpaCmd ツールが表示するエラーコードです。

```
LPC_NO_ERROR            =     0, // 正常終了（ゼロ保証）
LPC_ERR_CLT_SIGN        = -1200, // クライアント署名処理エラー
LPC_ERR_CLT_END         = -1201, // クライアント終了処理エラー
LPC_ERR_CLT_CANCEL      = -1202, // 証明書選択キャンセル
LPC_ERR_CLT_ARG         = -1203, // クライアント引数エラー
LPC_ERR_SRV_MAKE        = -1210, // サーバー仮署名処理エラー
LPC_ERR_SRV_ARG         = -1211, // サーバー引数エラー
LPC_ERR_SRV_CERT        = -1212, // サーバー署名の証明書エラー
LPC_ERR_SRV_XML         = -1213, // サーバー応答 XML エラー
LPC_ERR_SRV_EMBED       = -1214, // サーバー埋め込みエラー
LPC_ERR_FIELD_ARG       = -1220, // 署名フィールド引数エラー
LPC_ERR_FIELD_XML       = -1221, // 署名フィールド XML エラー
LPC_ERR_SIGN_ARG        = -1230, // 署名付与引数エラー
LPC_ERR_SIGN_PDF        = -1231, // 署名付与 PDF ファイルエラー
LPC_ERR_SIGN_XML        = -1232, // 署名付与 XML エラー
LPC_ERR_SIGN_NO_CERT    = -1233, // 署名付与証明書未指定エラー
LPC_ERR_SIGN_NO_TS      = -1234, // 署名付与タイムスタンプ未指定エラー
LPC_ERR_VERIFY_ARG      = -1240, // 検証引数エラー
LPC_ERR_VERIFY_XML      = -1241, // 検証 XML エラー
LPC_ERR_VERIFY_NO_REPORT = -1242, // 検証結果レポートエラー
LPC_ERR_VERIFY_INDETERM = -1250, // 検証結果が不明
LPC_ERR_VERIFY_INVALID  = -1251, // 検証結果が不正
LPC_ERR_VERIFY_NOT_LTV  = -1252, // 検証結果が LTV では無い
LPC_ERR_PDF_ARG         = -1260, // PDF 引数エラー
LPC_ERR_LTV_XML_OPEN    = -1270, // 検証結果 XML ファイルが開けない
LPC_ERR_READ_FILE       = -1300, // LpaCmd ファイル読み込みエラー
LPC_ERR_WRITE_FILE      = -1301, // LpaCmd ファイル書き込みエラー
LPC_ERR_NOT_SUPPORT     = -1390, // LpaCmd 未対応機能
LPC_ERR_SYSTEM          = -1397, // システムエラー（インスタンス生成失敗等）
LPC_ERR_EXCEPTION       = -1398, // 不明例外発生
LPC_ERR_UNKNOWN         = -1399, // 不明エラー
```

Ａ．３．LePDF エラーコード（src/LePDF/ILePDF.h）-2000 ～ -2499

LePDF のエラーコードは PDF ファイルの基礎的な解析や利用時に生じるエラーです。この範囲のエラーコードを生じた場合には PDF ファイルの形式異常の可能性があります。

```
LP_EXIST                =      1, // 既に存在している
LP_NO_ERROR             =      0, // 正常終了（ゼロ保証）
LP_ERR_ARGUMENT         = -2000, // 引数異常
LP_ERR_NOT_INIT         = -2001, // 未初期化
// base
LP_ERR_LOAD_FILE        = -2010, // 空のファイルか読み込みに失敗した
LP_ERR_NO_BUUFER        = -2011, // 中間バッファの取得に失敗した
LP_ERR_NO_PDF_FILE      = -2012, // ストリームの内容が PDF ファイルでは無い
LP_ERR_INVALID_PDF      = -2013, // PDF 形式の異常
LP_ERR_NO_EOF           = -2014, // EOF が見つからない
LP_ERR_NO_STARTXREF     = -2015, // startxref が見つからない
LP_ERR_OBJ_NOTFOUND     = -2016, // 指定オブジェクトが見つからない
LP_ERR_FILTER           = -2017, // 指定フィルターが見つからない
LP_ERR_NO_OBJ           = -2018, // オブジェクトが無い
LP_ERR_INVALID_OBJ      = -2019, // オブジェクト形式が異常
LP_ERR_NO_ROOT          = -2020, // root(catalog)が見つからない
LP_ERR_ROOT             = -2021, // root(catalog)のエラー
LP_ERR_OBJ_STREAM       = -2022, // stream の取得エラー
LP_ERR_OBJ_SAVE         = -2023, // オブジェクトの保存エラー
LP_ERR_XREF_SAVE        = -2024, // xref の保存エラー
LP_ERR_TRAILER_SAVE     = -2025, // Trailer の保存エラー
LP_ERR_SAVE_WRITER      = -2026, // 保存クラスエラー（バッファ不足等）
LP_ERR_METADATA         = -2027, // メタデータエラー
LP_ERR_EMBEDDED         = -2028, // 添付ファイルエラー
// data
LP_ERR_DATA_DICTKEY     = -2230, // 辞書の解析に失敗した
// parse
LP_ERR_PARSE_XREF       = -2040, // xref の解析に失敗した
LP_ERR_PARSE_TRAILER    = -2041, // trailer の解析に失敗した
LP_ERR_PARSE_DICT       = -2042, // 辞書の解析に失敗した
LP_ERR_PARSE_OBJ        = -2043, // オブジェクトの解析に失敗した
LP_ERR_PARSE_STREAM     = -2044, // ストリームの解析に失敗した
LP_ERR_PARSE_FILTER     = -2045, // フィルターの解析に失敗した
LP_ERR_PARSE_ENTRY      = -2046, // 不明なオブジェクトエントリーだった(Xref)
LP_ERR_PARSE_PARAMS     = -2047, // 辞書パラメータ解析エラー
LP_ERR_PARSE_OUTLINE    = -2048, // しおりの解析エラー
LP_ERR_PARSE_PAGE       = -2049, // ページの解析エラー
LP_ERR_PARSE_EMBED      = -2050, // ファイル埋め込みの解析エラー
LP_ERR_PARSE_CMAP       = -2051, // CMap の解析に失敗した
LP_ERR_PARSE_FONT       = -2052, // Font の解析に失敗した
LP_ERR_PARSE_XREFTYPE   = -2053, // 異なる種類 xref が混在している
LP_ERR_PARSE_XMP        = -2054, // XMP の解析に失敗した

// font
LP_ERR_FONT_MAKE        = -2060, // Font の生成に失敗した
// page
LP_ERR_PAGE_ROTATE      = -2070, // ページ回転エラー
```

```
LP_ERR_PAGE_MEDIABOX     = -2071, // ページ MediaBox エラー
LP_ERR_PAGE_RSRC         = -2072, // ページリソースエラー
// pages/outlines
LP_ERR_PAGES_NOTINIT     = -2080, // Pages 解析がされていない
LP_ERR_OLINES_NOTINIT    = -2081, // Outlines 解析がされていない
LP_ERR_EMBED_NOTINIT     = -2082, // EmbeddedFiles 解析がされていない
// encrypt
LP_ERR_ENC_PASSWD        = -2100, // パスワードエラー
LP_ERR_PARSE_KEY         = -2101, // 復号鍵計算時のエラー
LP_ERR_NO_ENCRYPT_INFO   = -2102, // Encrypt 情報が無いか誤っている
LP_ERR_ENC_NO_FILTER     = -2103, // 暗号辞書 Filter が無い（必須）
LP_ERR_ENC_INVALID_VNUM  = -2104, // 暗号辞書 V 値エラー
LP_ERR_ENC_UNKNOWN_VNUM  = -2105, // 暗号辞書 V 値エラー（未サポートの可能性あり）
LP_ERR_ENC_FILTER        = -2106, // 暗号辞書 Filter のエラー
LP_ERR_ENC_R             = -2107, // 暗号辞書 R のエラー（必須）
LP_ERR_ENC_O             = -2108, // 暗号辞書 O のエラー（必須）
LP_ERR_ENC_U             = -2109, // 暗号辞書 U のエラー（必須）
LP_ERR_ENC_P             = -2110, // 暗号辞書 P のエラー（必須）
LP_ERR_ENC_STMF          = -2111, // 暗号辞書 StmF のエラー
LP_ERR_ENC_STRF          = -2112, // 暗号辞書 StrF のエラー
LP_ERR_ENC_EFF           = -2113, // 暗号辞書 EFF のエラー
// inflate
LP_ERR_INFLATE_INIT      = -2120, // inflate 初期化時のエラー
LP_ERR_INFLATE_ERROR     = -2121, // inflate のエラー
LP_ERR_INFLATE_BUFFER    = -2122, // 解凍用のメモリ不足エラー
// zlib
LP_ERR_ZLIB_INIT         = -2140, // zlib 初期化時のエラー
LP_ERR_ZLIB_ERROR        = -2141, // zlib 解凍時のエラー
// Crypto(HASH)
LP_ERR_MD5_ERROR         = -2200, // MD5 ダイジェスト計算時のエラー
LP_ERR_SHA_ERROR         = -2201, // SHA-1/2 ダイジェスト計算時のエラー
// Crypto(ENCRYPT)
LP_ERR_RC4_INIT          = -2220, // RC4 暗号初期化時のエラー
LP_ERR_RC4_USE           = -2221, // RC4 暗号復号時のエラー
LP_ERR_RC4_KEY           = -2222, // RC4 鍵のエラー
LP_ERR_AES_ERROR         = -2223, // AES 暗号計算時のエラー
// その他
LP_ERR_NO_SUPPORTED      = -2490, // 現在未サポートの機能が使われた
LP_ERR_SYSTEM            = -2497, // システムエラー（インスタンス生成失敗等）
LP_ERR_EXCEPTION         = -2498, // 不明例外発生
LP_ERR_UNKNOWN           = -2499, // 不明エラー
```

Ａ．４．LePKI エラーコード（include/LePKI/LePKI.h）-3000 ～ -3999

LePKI のエラーコードは PKI（公開鍵基盤）の操作に関するエラーで生じます。

```
LPK_NO_ERROR            =     0, // 正常終了（ゼロ保証）
LPK_ERR_ARGUMENT        = -3000, // 引数異常
LPK_ERR_NOT_INIT        = -3001, // 未初期化
// LePKI
LPK_ERR_PKI_INIT        = -3100, // LePKI 未初期化
LPK_ERR_WINSTORE        = -3101, // Windows 証明書ストアエラー
LPK_ERR_GETMODULE       = -3102, // 実行ファイルパス取得エラー
LPK_ERR_SET_STORE       = -3103, // 証明書ストアセットエラー
LPK_ERR_SET_CACHE       = -3104, // CRL キャッシュセットエラー
LPK_ERR_ADD_REPOSIT     = -3105, // リポジトリサーバセットエラー
LPK_ERR_ADD_CVS         = -3106, // 証明書検証サーバセットエラー
LPK_ERR_ADD_BASICAUTH   = -3107, // 基本認証セットエラー
LPK_ERR_PATH_BUILD      = -3108, // 認証パス構築エラー
LPK_ERR_PATH_VALID      = -3109, // 認証パスの検証情報収集エラー
LPK_ERR_PATH_VERIFY     = -3110, // 認証パスの検証エラー
LPK_ERR_GET_CRL         = -3111, // CRL 取得エラー
LPK_ERR_GET_PARENT      = -3112, // 親証明書取得エラー
// LpkCert
LPK_ERR_WCERT_SETFILE   = -3200, // Windows 証明書ファイルセットエラー
LPK_ERR_WCERT_SETP12    = -3201, // Windows 証明書(PKCS#12)セットエラー
LPK_ERR_WCERT_NOTFOUND  = -3202, // Windows 証明書未定義エラー
LPK_ERR_WCERT_CANCEL    = -3203, // Windows 証明書選択キャンセル
LPK_ERR_WCERT_CAPI      = -3204, // Windows 証明書 CAPI エラー
LPK_ERR_OCERT_SETFILE   = -3210, // OpenSSL 証明書ファイルセットエラー
LPK_ERR_OCERT_SETBIN    = -3211, // OpenSSL 証明書バイナリセットエラー
LPK_ERR_OCERT_SETP12    = -3212, // OpenSSL 証明書(PKCS#12)セットエラー
LPK_ERR_OCERT_INITP11   = -3215, // OpenSSL 証明書(PKCS#11)初期化エラー
LPK_ERR_OCERT_SIGNP11   = -3216, // OpenSSL 証明書(PKCS#11)署名エラー
LPK_ERR_CERT_GET_INFO   = -3220, // 証明書からの情報取得エラー(情報が無い場合を含む)
// LpkCrl
LPK_ERR_CRL_INIT        = -3230, // LpkCrl 未初期化
LPK_ERR_OCRL_CHECK      = -3231, // LpkCrl(OpenSSL)確認エラー
// LpkOcsp
LPK_ERR_OCSP_INIT       = -3240, // LpkOcsp 未初期化
LPK_ERR_OOCSP_STATUS    = -3241, // LpkOcsp(OpenSSL)確認エラー
LPK_ERR_OOCSP2_STATUS   = -3242, // LpkOcsp(LeBerXml)確認エラー
// LpkCades
LPK_ERR_CADES_INIT      = -3250, // LpkCades 未初期化
LPK_ERR_CADES_ADDTS     = -3251, // LpkCades タイムスタンプ追加エラー
LPK_ERR_CADES_SIGN      = -3252, // LpkCades 署名エラー
LPK_ERR_CADES_VERIFY    = -3253, // LpkCades 検証エラー
LPK_ERR_CADES_HASH      = -3254, // LpkCades ハッシュ方式エラー
LPK_ERR_CADES_SETTS     = -3255, // LpkCades タイムスタンプセットエラー
LPK_ERR_CADES_SETBIN    = -3256, // LpkCades バイナリセットエラー
LPK_ERR_CADES_SETREVO   = -3257, // LpkCades 失効情報セットエラー
// LpkPkcs7
LPK_ERR_PKCS7_INIT      = -3270, // LpkPkcs7 未初期化
LPK_ERR_PKCS7_ADDTS     = -3271, // LpkPkcs7 タイムスタンプ追加エラー
```

```
LPK_ERR_OPKCS7_SIGN      = -3272, // LpkPkcs7(OpenSSL)署名エラー
LPK_ERR_OPKCS7_VERIFY    = -3273, // LpkPkcs7(OpenSSL)検証エラー
LPK_ERR_OPKCS7_HASH      = -3274, // LpkPkcs7(OpenSSL)ハッシュ方式エラー
LPK_ERR_OPKCS7_SETTS     = -3275, // LpkPkcs7(OpenSSL)タイムスタンプセットエラー
LPK_ERR_OPKCS7_SETBIN    = -3276, // LpkPkcs7(OpenSSL)PKCS#7 バイナリセットエラー
// LpkTimestampToken
LPK_ERR_TST_INIT         = -3290, // LpkTimestampToken 未初期化
LPK_ERR_OTST_INIT        = -3291, // LpkTimestampToken(OpenSSL)未初期化
LPK_ERR_OTST_MSGIMPL     = -3292, // LpkTimestampToken(OpenSSL)ハッシュ値取得エラー
// LpkTimestamp
LPK_ERR_TS_PARSE         = -3300, // LpkTimestamp 結果解析エラー
LPK_ERR_TS_SET           = -3301, // LpkTimestamp 設定エラー
// LpkTsAmano
LPK_ERR_ATS_SET          = -3320, // LpkTsAmano 設定エラー
// LpkUtil
LPK_ERR_UTIL_GETCRL      = -3350, // LpkUtil の CRL 取得エラー
LPK_ERR_UTIL_GETOCSP     = -3351, // LpkUtil の OCSP 取得エラー
LPK_ERR_UTIL_GETCERT     = -3352, // LpkUtil の証明書取得エラー
LPK_ERR_UTIL_GETCVS      = -3353, // LpkUtil の CVS 取得エラー
LPK_ERR_UTIL_GETOCSP2    = -3354, // LpkUtil の独自 OCSP 取得エラー
LPK_ERR_UTIL_HTTP        = -3355, // HTTP 通信エラー
// LpkCrypto
LPK_ERR_HASH_INIT        = -3400, // ハッシュ初期化エラー
LPK_ERR_HASH_ERROR       = -3401, // ハッシュエラー
LPK_ERR_ENC_INIT         = -3410, // 共通鍵暗号初期化エラー
LPK_ERR_ENC_ERROR        = -3411, // 共通鍵暗号エラー
LPK_ERR_PKEY_INIT        = -3420, // 公開鍵暗号初期化エラー
LPK_ERR_PKEY_ERROR       = -3421, // 公開鍵暗号エラー
LPK_ERR_PKEY_GET         = -3422, // 公開鍵暗号利用エラー
LPK_ERR_SIGN_ALG         = -3430, // 不明な公開鍵暗号アルゴリズム
// LeBerXml 用確保
// -3600 ～ -3699 の定義は LeBerXml/LeBerXml.h の中で定義
// XML
LPK_ERR_PKI_XML_LOAD     = -3700, // LePKI の XML 読み込みエラー
LPK_ERR_PKI_XML_SAVE     = -3701, // LePKI の XML 書き込みエラー
LPK_ERR_CERT_XML_LOAD    = -3710, // LpkCert の XML 読み込みエラー
LPK_ERR_CERT_XML_SAVE    = -3711, // LpkCert の XML 書き込みエラー
LPK_ERR_TS_XML_LOAD      = -3720, // LpkTimestamp の XML 読み込みエラー
LPK_ERR_TS_XML_SAVE      = -3721, // LpkTimestamp の XML 書き込みエラー
// JNI
LPK_ERR_JNI_ARGUMENT     = -3800, // LePKI の JNI 引数エラー
LPK_ERR_JNI_INIT         = -3801, // LePKI の JNI 初期化エラー
LPK_ERR_JNI_EXEC         = -3802, // LePKI の JNI の API エラー
// .NET
LPK_ERR_DOTNET_ARGUMENT  = -3810, // LePKI の.NET 引数エラー
LPK_ERR_DOTNET_INIT      = -3811, // LePKI の.NET 初期化エラー
LPK_ERR_DOTNET_EXEC      = -3812, // LePKI の.NET の API エラー
// その他
LPK_ERR_NO_SUPPORTED     = -3990, // 現在未サポートの機能が使われた
LPK_ERR_SYSTEM           = -3997, // システムエラー（インスタンス生成失敗等）
LPK_ERR_EXCEPTION        = -3998, // 不明例外発生
LPK_ERR_UNKNOWN          = -3999, // 不明エラー
```

Ａ．５．LePAdES エラーコード（include/LePAdES/LePAdES.h）-4000 ～ -4999

LePAdES のエラーコードは PDF 電子署名に関するエラーです。

```
PDA_NO_ERROR                    =     0, // 正常終了（ゼロ保証）
PDA_ERR_ARGUMENT                = -4000, // 引数異常（通常はあり得ない）
PDA_ERR_NOT_INIT                = -4001, // 未初期化（通常はあり得ない）
// 解析
PDA_ERR_PARSE_FIELDS            = -4100, // AcroForm の Fields 異常
PDA_ERR_PARSE_SIGDICT           = -4101, // SigDict の異常
PDA_ERR_PARSE_DSS               = -4102, // DSS 辞書の異常
PDA_ERR_PARSE_VRI               = -4103, // VRI 辞書の異常
// 生成
PDA_ERR_ADD_FIELD               = -4200, // 署名フィールドの追加エラー
PDA_ERR_MAKE_FIELD              = -4201, // 署名フィールド生成エラー
PDA_ERR_MAKE_SIGDICT            = -4202, // 署名辞書生成エラー
PDA_ERR_MAKE_SIGN               = -4203, // 署名生成エラー
PDA_ERR_FIELD_ALREADY_EXIST     = -4204, // 署名フィールドが既に存在している
PDA_ERR_UPDATE_BYTERANGE        = -4220, // バイトレンジ更新エラー
PDA_ERR_UPDATE_SIGNDATA         = -4221, // 署名データ更新エラー
PDA_ERR_FIELD_NAME_NOTFOUND     = -4230, // 署名フィールド名が見つからない
PDA_ERR_SIGDICT_NOTFOUND        = -4231, // 署名辞書が見つからない
PDA_ERR_UNKNOWN_SUBFILTER       = -4232, // 不明なサブフィルター種類
PDA_ERR_FIELD_ALREADY_SIGNED    = -4233, // 指定された署名フィールド名は既に署名済み
PDA_ERR_GET_BYTERANGE           = -4240, // バイトレンジ取得エラー
PDA_ERR_GET_FIELD               = -4241, // 署名フィールド取得エラー
PDA_ERR_GET_SIGDICT             = -4242, // 署名辞書取得エラー
PDA_ERR_GET_TARGET              = -4243, // 署名対象取得エラー
PDA_ERR_GET_DOCTS               = -4244, // ドキュメントタイムスタンプ取得エラー
PDA_ERR_GET_CONTENTS            = -4250, // 署名データ取得エラー
PDA_ERR_SET_CONTENTS            = -4251, // 署名データセットエラー
PDA_ERR_BYTERANGE               = -4260, // 異常なバイトレンジ値だった
PDA_ERR_CONTENTS_BUF            = -4261, // 確保済み署名データ領域が不足
// 検証（詳細な検証エラーは検証結果 XML として出力される）
PDA_ERR_VERIFY_CRYPTO           = -4300, // 暗号検証エラー
PDA_ERR_VERIFY_CERTS            = -4301, // 証明書検証エラー
PDA_ERR_VERIFY_TIMESTAMP        = -4302, // タイムスタンプ検証エラー
// 外観
PDA_ERR_APEA_MAKE               = -4400, // 外観生成エラー
PDA_ERR_APEA_FIELD_NOTFOUND     = -4401, // 署名フィールドが見つからない
PDA_ERR_APEA_NO_TEXT            = -4402, // 表示するテキストが無い
PDA_ERR_APEA_DOCTIMESTAMP       = -4403, // ドキュメントタイムスタンプに許されていない外観
PDA_ERR_APEA_XOBJECT            = -4404, // 外観リソース生成エラー
PDA_ERR_APEA_BLANK              = -4405, // ブランク外観生成エラー
PDA_ERR_APEA_MAKE_FONT          = -4420, // フォント生成エラー
PDA_ERR_APEA_MAKE_GRAP          = -4421, // 画像情報生成エラー
PDA_ERR_APEA_MAKE_RSRC_OBJ      = -4422, // 画像リソース生成エラー
PDA_ERR_APEA_MAKE_IMAGE         = -4423, // 画像データ生成エラー
PDA_ERR_APEA_MAKE_IMAGEFILE     = -4424, // 画像ファイル読み込みエラー
PDA_ERR_APEA_MAKE_PDF_FILE      = -4425, // 外観 PDF ファイル読み込みエラー
PDA_ERR_APEA_MAKE_RSRC          = -4426, // リソース取得エラー
PDA_ERR_APEA_MEDIA_BOX          = -4430, // 領域取得エラー
```

```
PDA_ERR_APEA_STREAM                = -4440, // オペレータ生成エラー
PDA_ERR_APEA_N2                    = -4441, // N2 オブジェクト生成エラー
PDA_ERR_APEA_2ND                   = -4442, // 2nd オブジェクト生成エラー
PDA_ERR_APEA_TOP                   = -4443, // TOP オブジェクト生成エラー
PDA_ERR_APEA_FILE_NOTFOUND         = -4450, // 外観指定ファイルが開けない
PDA_ERR_APEA_PNG2BMP               = -4480, // PNG から BMP の変換エラー
// 長期署名
PDA_ERR_LTV_NOT_VALID              = -4500, // 検証結果が VALID では無い
PDA_ERR_LTV_XML_NOVALUE   = -4501, // 検証結果に値が無い(value オプションがオフか NET 未使用)
PDA_ERR_LTV_CANNOT_PARSE           = -4502, // 検証結果の解析に失敗した
// XML
PDA_ERR_PADES_XML_LOAD             = -4700, // LePAdES の XML 読み込みエラー
PDA_ERR_PADES_XML_SAVE             = -4701, // LePAdES の XML 書き込みエラー
PDA_ERR_APPEARANCE_XML_LOAD        = -4710, // PdaAppearance の XML 読み込みエラー
PDA_ERR_APPEARANCE_XML_SAVE        = -4711, // PdaAppearance の XML 書き込みエラー
PDA_ERR_FIELD_XML_LOAD             = -4720, // PdaField の XML 読み込みエラー
PDA_ERR_FIELD_XML_SAVE             = -4721, // PdaField の XML 書き込みエラー
PDA_ERR_SIGN_XML_LOAD              = -4730, // PdaSign の XML 読み込みエラー
PDA_ERR_SIGN_XML_SAVE              = -4731, // PdaSign の XML 書き込みエラー
PDA_ERR_PARSEXML_XML_LOAD          = -4740, // PdaParseXml の XML 読み込みエラー
PDA_ERR_PARSEXML_XML_SAVE          = -4741, // PdaParseXml の XML 書き込みエラー
PDA_ERR_VERIFYXML_XML_LOAD         = -4750, // PdaVerifyXml の XML 読み込みエラー
PDA_ERR_VERIFYXML_XML_INIT         = -4751, // PdaVerifyXml の XML 未初期化エラー
PDA_ERR_VERIFYXML_XML_STAUS        = -4752, // PdaVerifyXml の XML 結果種別エラー
// JNI
PDA_ERR_JNI_ARGUMENT               = -4800, // LePAdES の JNI 引数エラー
PDA_ERR_JNI_INIT                   = -4801, // LePAdES の JNI 初期化エラー
// .NET
PDA_ERR_DOTNET_ARGUMENT            = -4810, // LePAdES の.NET 引数エラー
PDA_ERR_DOTNET_INIT                = -4811, // LePAdES の.NET 初期化エラー
// 入出力
PDA_ERR_SAVE_BUFFER                = -4850, // バッファ保存エラー
PDA_ERR_ALREADY_SIGNED             = -4851, // 署名済みなので操作出来ない
// その他
PDA_ERR_NO_SUPPORTED               = -4990, // 現在未サポートの機能が使われた
PDA_ERR_NO_ENHANCED       = -4991, // LE:PAdES-Basic:Lib で PAdES-Enhanced の機能が使われた
PDA_ERR_EVALUATION                 = -4992, // 評価版の制限エラー
PDA_ERR_SYSTEM                     = -4997, // システムエラー（インスタンス生成失敗等）
PDA_ERR_EXCEPTION                  = -4998, // 不明例外発生
PDA_ERR_UNKNOWN                    = -4999, // 不明エラー
```

付録Ｂ．制限事項・履歴

- V1.08 では現在以下の制限がある。

  1）署名外観のフォントは現在 "MS-Mincho" のみ利用可能です。
  2）32bit 版では 200 メガバイトを超えるような大きな PDF ファイルの利用は未対応です。
  3）証明書による暗号化には未対応です。
  4）PdaEncrypt（新規暗号化）クラスは未サポートです。
  5）その他未対応の機能には API リファレンスにて「(V1.0 では未サポート)」と記述。

- その他バージョンアップの履歴に関してはルートディレクトリ下にある readme-LePAdES.txt に記載。

- V1.08 では２つの仕様変更があった。詳しくは 8 ページの「Ver1.08.R1 の仕様変更に関する注意点」を参照。

付録Ｃ．ＦＡＱ（よくある質問）

Ｃ．１．仕様について

Q	長期署名とは何ですか？
A	証明書の有効期限を超えて署名を有効（検証可能）にする仕組みです。通常の電子署名では証明書の有効期限後には正しく検証できなくなります。証明書の有効期限は通常数年ですので 5～10 年以上の保管と有効性を保つ用途には長期署名が必要となります。逆に長期保管が必要無い場合には長期署名では無く通常の電子署名で構いません。長期署名に関して詳しくは開発元ラング・エッジの「長期署名とは？」をご覧ください。 http://www.langedge.jp/biz/AdES.html

Q	長期署名（ISO32000-2/PAdES 仕様）と従来の電子署名（ISO32000-1/PDF 署名）とは何が異なりますか？
A	PDF の長期署名（PAdES）仕様は ISO32000-2 から採用されました。従来の ISO32000-1（PDF 署名）仕様と比較して以下の 3 つの要素が追加されました。 1. CAdES を署名データに利用した、PAdES-Enhanced 仕様 2. 長期署名用に証明書と検証情報を埋め込む、PAdES-LTV の DSS/VRI 仕様 3. タイムスタンプだけを付与できる、PAdES-LTV の DocTimestamp 仕様 一方で従来の ISO32000-1（PDF 署名）仕様は、PAdES-Basic 仕様として再整理されました。『長期署名 PAdES ライブラリ』では機能レベルにより 2 製品が提供されています。

Q	ISO32000-2 の PAdES 仕様と ETSI（欧州電気通信標準化機構）の PAdES 仕様の関係はどうなっていますか？
A	長期署名の仕様は元々EU（欧州）の標準化組織である ETSI の TC ESI（電子署名基盤技術委員会）にて標準化されました。XML と CMS の長期署名仕様である、XAdES と CAdES に続いて、PDF の長期署名仕様である PAdES に関しても ETSI TS 102 778 として公開されました。PDF の標準仕様である ISO32000-2（現在標準化作業中で 2014～2015 年に完了予定）は、ETSI TS 102 778 の PAdES 仕様が反映されています。つまり PAdES の元仕様は ETSI の PAdES 仕様であり、後から ISO32000 に反映される関係となります。

Q	PDF/A 仕様をサポートしていますか？
A	『長期署名 PAdES ライブラリ』は、PDF/A をサポートしていません。PDF/A 準拠の PDF に署名はできますが、署名した PDF ファイルは PDF/A 準拠となりません。

Q	PDF の暗号化をサポートしていますか？
A	未暗号化の入力ファイルに対しての暗号化はできませんが、暗号化済みの入力ファイルへの署名は可能です。暗号方式としてパスワードによる暗号化に関しては Acrobat-9 までの暗号化仕様に対応しています。証明書（デジタル ID）による暗号化には対応しておりません。

C．２．証明書/タイムスタンプについて

Q	IC カードや USB トークンに格納された証明書（秘密鍵）を署名に使うことができますか？
A	Windows 版では、CryptoAPI（CAPI）対応の USB トークンや IC カードであれば署名に利用できます。Linux 版では基本的に対応していません。実績等詳しくは「４．１．証明書」をご覧ください。

Q	RSA-2048bit・SHA-2 の証明書（秘密鍵）に対応していますか？
A	対応しています。RSA は 2048bit と 4096bit に、ハッシュ方式は SHA-1 と SHA-2（256bit/384bit/512bit）がご利用頂けます。

Q	利用実績のある認証局（証明書発行元）を教えてください。
A	基本的には PKCS#12 ファイル形式または Windows 版であれば CAPI 利用の IC カード等で証明書を提供している認証局であれば利用が可能です。GPKI 等の官職証明書や IC カードも利用可能です。実績等詳しくは「４．１．証明書」をご覧ください。

Q	OpenSSL 等を使って自分で生成した証明書と秘密鍵は使えますか？
A	PKCS#12 形式のファイルにすれば利用可能です。Windows 版では PKCS#12 以外に Windows 証明書ストアにインストール頂いても利用頂けます。

Q	Adobe CDS 証明書に対応していますか？
A	対応していないので、ご利用頂けません。Adobe CDS（Certified Document Service）証明書は Adobe 社認定の認証局でのみ利用が可能であり、サードベンダーには利用が許可されていないためです。

Q	利用実績のある商用タイムスタンプサービス（認定 TSA）を教えてください。
A	国内の主な商用タイムスタンプサービス（アマノタイムスタンプサービス 3161 や SEIKO 認定タイムスタンプサービス等）には標準で対応しています。実績等詳しくは「４．２．タイムスタンプ」をご覧ください。

Q	認証局や電子署名の勉強をするのに良い資料はありませんか？
A	認証局の業界組織である電子認証局会議にて公開されている「電子署名活用ガイド」がおすすめです。法的な観点からだけでなく事例まで分かりやすくまとまっています。冊子としても配布されていますが、PDF ファイルとしてダウンロードも可能です。 http://www.c-a-c.jp/download/index.html

Q	PFX 形式の証明書（秘密鍵）ファイルとは、PKCS#12 形式のファイルと同じですか？
A	同じと考えて頂いて問題ありません。PFX（Personal inFormation eXchange）は PKCS#12 の旧称となります。Windows 環境では拡張子 .pfx と .p12 のどちらでも構いません。

C．3．API/コマンドラインについて

Q	JAVA API が対応している JDK のバージョンを教えてください。
A	『長期署名 PAdES ライブラリ』は、JDK 1.6（JRE 6.0）以降でお使い頂けます。なお JNI を利用しているので 32bit 版と 64bit 版は『長期署名 PAdES ライブラリ』と JAVA の JDK/JRE で一致している必要があります。

Q	JAVA のキーストアは利用できますか？
A	利用できません。『長期署名 PAdES ライブラリ』の JAVA API は JNI（Java からのネイティブ呼び出し）を利用しているので、JAVA 環境のキーストアにはアクセスできません。JAVA API からも利用でも、PKCS#12 ファイル形式または Windows 版であれば Windows 証明書ストアを利用してください。なお JAVA API のサンプルが sample/LePAdES/java の下に用意されています。

Q	Windows 版 C++ API が対応している Visual Studio のバージョンを教えてください。
A	標準で Visual Studio 2010 に対応しています。再ビルドすればその他の環境でも利用可能ですのでご相談ください。なお C++ API のサンプルが sample/LePAdES/cpp の下に用意されています。

Q	コマンドラインで PKCS#12 ファイルを利用して署名する方法を教えてください。
A	以下のコマンドラインにて、不可視署名で署名フィールド名 "SIGN1" へ署名することができます。PKCS#12 ファイルとパスワードが必要です。コマンドラインのサンプルが sample/LePAdES/cmd の下に用意されています。 例＞ $ LpaCmd -sign -newf -cert p12 test.p12 passwd -in input.pdf -out singed.pdf

Q	コマンドラインで署名済み PDF ファイルを検証する方法を教えてください。
A	以下のコマンドラインにて、検証を行い結果（レポート）を画面に表示することができます。コマンドラインのサンプルが sample/LePAdES/cmd の下に用意されています。 例＞ $ LpaCmd -verify -report -in singed.pdf

C．4．署名機能について

Q	IC カードや USB トークンに格納された証明書（秘密鍵）の署名時に PIN コードの入力画面が開いてしまいます。API やコマンドラインの引数に PIN コードをセットして PIN コードの入力画面を開かず、一括で署名処理を行うことは可能でしょうか？
A	できません。PIN コードの入力画面は IC カードや USB トークンのドライバが表示していますので、『長期署名 PAdES ライブラリ』の API では対応できません。

Q	署名外観の印影を透過（スタンプの下の文字や文面が見える状態）にしたいのですが、どのように設定したら良いでしょうか？
A	印影として画像ファイルを指定した場合には透過にはできません。透過 PNG も透過にはなりません。透過にしたい場合には印影として PDF ファイルを指定してください。この場合 PDF ファイルにはベクトルにより印影をセットします。

Q	署名形式で指定する署名データ形式のうち、PKCS#7 と CAdES の違いを教えてください。
A	PKCS#7 は CAdES のベースともなった署名データ形式ですので似ていますが、署名証明書を保護する属性等が CAdES では追加されているので、CAdES の方がよりセキュアな署名データ形式と言えます。今後生成する署名では可能な限り CAdES 署名データ（PAdES-Enhanced）の利用をおすすめします。

Q	署名種別で指定する、普通署名と MDP 署名の違いを教えてください。
A	複数の署名を付与して行く場合には普通署名を使います。MDP 署名時にはその後可能な操作の制限ができます。通常 MDP 署名の後では署名の追加はできなくなります。

Q	証明書の署名アルゴリズムに使われているハッシュ方式（例：SHA-256）と、署名時に指定するハッシュ方式は同じである必要がありますか？
A	証明書の署名と、PDF 署名/PAdES の署名は全く別のデータですので両者のハッシュ方式等は異なっていても構いません。SHA-1 署名の証明書で、SHA-2 署名の PDF 署名を付与することも可能ですし、その逆も可能です。

Q	『長期署名 PAdES ライブラリ』で作成した署名フィールドに、Adobe Acrobat/Reader で電子署名を付ける事はできますか？
A	Adobe Acrobat であれば電子署名を付けることができますが、Adobe Reader では電子署名を付けることはできません。Adobe Reader で電子署名を付けるには、サードベンダーには許可されていない方法（つまり Adobe 製品でのみ可能な方法）で署名フィールドを生成しておく必要があります。

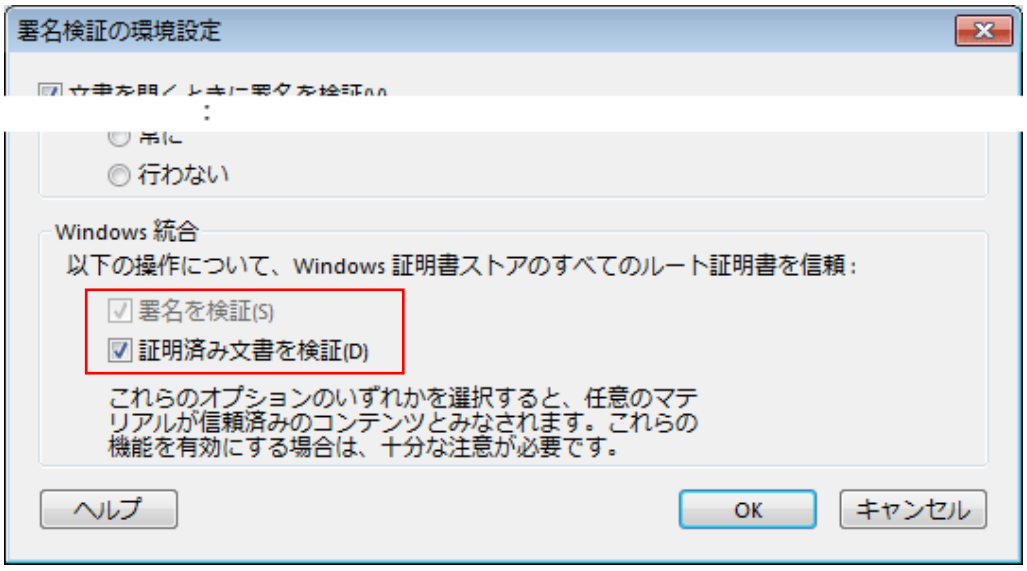
Ｃ．５．検証機能について

Q	Adobe Acrobat/Reader と検証結果は一致しますか？
A	ほぼ一致しますが相互の設定や対応状況の違いにより一致しないケースもあります。例えばX500名とディレクトリサーバによる検証が必要な証明書（GPKI の官職証明書等）は Adobe Acrobat/Reader では標準対応していませんし、毎年ルート証明書が更新して、同時に複数の有効なルート証明書がある証明書（商業登記証明書等）では、Adobe Acrobat/Reader では検証できない時があります。『長期署名 PAdES ライブラリ』ではどちらも対応しています。

Q	署名した PDF を Acrobat で確認したところ、「有効な証明済み」とはなっているのですが、「署名は有効ですが、署名者の ID が失効しているかどうかを確認できませんでした」という説明がついてしまいました。これはどうすれば解決できるのでしょうか？
A	これは『長期署名 PAdES ライブラリ』の問題ではなく、CRL（失効した証明書のリスト）または OCSP（証明書の失効問合せ）の取得に失敗している可能性が高いです。ネット環境として CRL 配布ポイントや OCSP サーバにアクセス可能かどうかを確認してください。特に取得に LDAP を使っている場合にはファイアウォール等で許可されていない場合があります。他にも認証局のサーバが一時的にメンテナンスを行なっている場合にもこのエラーが出ることがあります。この場合は、時間をおいて試すことで成功します。

Q	自分で生成した証明書（秘密鍵）で署名を行い、作成された PDF を別マシンに持っていた際に、証明書が無いため、署名の有効性が不明となります。この場合、署名自体に信頼がないとしても、PDF が変更されていないことが保証されていると考えてよいでしょうか？
A	無効の理由として「署名後、文書が変更されたか壊れています」と出なければ改変されていないと言う事になります。ただし秘密鍵が流出して偽造されたかどうかの確認は、署名者が確認でき無いので保証できません。

Q	Adobe Acrobat/Reader で複数の署名を検証すると最初の署名の後に「その他の変更が 1 個あります」と表示されますが、『長期署名 PAdES ライブラリ』の検証結果には表示されません。この表示の意味は何でしょうか？また『長期署名 PAdES ライブラリ』でもこの変更を判断できますか？
A	署名後に署名を追加した、と言うことが「その他の変更」として表示されます。つまり署名した後に何か追加した場合に表示されます。増分更新と言う仕組みで追加していますので、その前の署名自体は正しく検証できます。 『長期署名 PAdES ライブラリ』でこの増分更新をチェックするには LePAdES::getSign() で署名辞書の情報を取得して、PdaSign::getByteRange() で署名範囲（ByteRange）を取得して判断します。ByteRange では［開始位置 1　長さ 1　開始位置 2　長さ 2］の 4 つの数値が取得できます。このうち「開始位置 2＋長さ 2」がその署名のサイズです。ファイルサイズがこの値よりも大きい場合には、署名後に追加（その他の変更）があったと判断できます。

Q	署名した PDF を Windows 環境上の Adobe Acrobat で確認したところ、「署名の完全性は不明です」となっていました。また、その説明として「署名者の ID は信頼済み証明書の一覧に見つからず、親証明書も信頼済み証明書ではないため不明です」とありました。どうすれば正常に認識されるのでしょうか？
A	署名証明書かルート証明書が信頼されていません。次の三通りの方法があります。 1. 自分で信頼する証明書に登録する。 たとえば Adobe Acrobat/Reader の「署名のプロパティ」ダイアログから「概要」タブ の「証明書を表示」をクリックすると、「証明書ビューア」ダイアログが開きます。ここの「信頼」タブで「信頼済み証明書に追加」の設定をします。 ただし、これは利用する環境すべてで設定する必要があります。 2. 信頼されているパブリック認証局から証明書を購入する（有償です） WebTrust 認定されたパブリック認証局であれば標準で Windows 証明書ストアにルート証明書が信頼済みとなっています。この場合には Adobe Acrobat/Reader に Windows 証明書ストアを利用する設定が必要です。 ＞ Adobe Acrobat/Reader 9 以前であれば、「環境設定」ダイアログから「セキュリティ」の「詳細環境設定」をクリックすると「電子署名の詳細環境設定」ダイアログが開きます。そこの「Windows 統合」タブのチェックを全てオンにしておけば信頼されます。 ＞ Adobe Acrobat/Reader X 以降であれば、「環境設定」ダイアログから「署名」タブの「検証」をクリックすると「署名検証の環境設定」ダイアログが開きます。そこの「Windows 統合」のチェックを全てオンにしておけば信頼されます。  デフォルト設定は Windows 証明書ストアを利用しない状態になっているため、この設定が必要となります。 3. その他 使用している証明書のルート証明書が標準で Windows 環境に入っていない場合は、必要なルート証明書を取得して自分で Windows 証明書ストアの「信頼されたルート証明書機関」にインポートする必要があります。Windows 証明書ストアの証明書は、プロパティの「インターネット オプション」の「コンテンツ」タブから「証明書」をクリックすることで画面を開くことができます。

Q	Adobe Reader/Acrobat で検証すると「署名検証中のエラー　この署名のフォーマットはこの署名方法ではサポートされていません。新しいバージョンの署名ハンドラが必要である可能性があります。」と表示されます。どうすれば良いでしょうか？
A	CAdES を使った電子署名（PAdES-Enhanced）または DocTimestamp（ドキュメントタイムスタンプ）を、未サポートの Adobe Reader/Acrobat 9 以前で開いています。Adobe Reader/Acrobat X 以降を利用して検証してください。

Q	検証に利用する CRL と OCSP とは何が違うのでしょうか？どちらを利用すれば良いのでしょうか？
A	CRL は失効した証明書のリストを使って失効していないことを確認する仕組み、OCSP は証明書の失効をサーバーに問合せして確認する仕組みです。どちらを使うかは認証局で決められており、普通は証明書に CRL や OCSP の情報が記載されています。従って自動的に CRL か OCSP を使って失効検証が行われますので、普通はどちらと指定する必要はありません。

Q	『長期署名 PAdES ライブラリ』で商用タイムスタンプを付けたのですが、Adobe Acrobat/Reader で署名を検証すると、時刻が不明になってしまいます。
A	この現象は、Acrobat/Adobe Reader でタイムスタンプ局の証明書の信頼性を検証できない状態になっている時に生じます。なお、Adobe Acrobat/Reader 7 では SHA-2 方式に未対応ですので、SHA-2 を利用している日本の商用タイムスタンプの検証はできませんので、ご注意ください。

C．６．その他（性能等）について

Q	何か制限があれば教えてください。
A	現在入力ファイルのサイズが 200MB に制限されています（カスタマイズ対応は可能です）。他にも細かい制限がありますが、詳しくは「付録B．制限事項」をご覧ください。

Q	何かオープンソースのプロジェクトやライブラリを利用していますか？またそれに伴うライセンス的な制限はありませんか？
A	幾つかのオープンソースを利用しています。利用しているのは、OpenSSL/ OpenLDAP/ libjpeg/ libpng/ libxml2/ zlib/ iconv/ AES/ MD5 です。いずれも商用利用可能かつソースコード公開の義務が無いライブラリです。コピーライト表示は必要なものがありますが、詳しくは製品マニュアルに記載されています。評価版を入手してご覧ください。

Q	署名を付与する場合にメモリサイズは性能に影響しますか？
A	大きな入力ファイルを同時に処理する場合にはメモリサイズが足りなくなり性能が低下する可能性があります。小さな入力ファイルを使っている場合はメモリサイズの影響はほとんどありません。

Q	署名を付与する場合の性能を教えてください。
A	以下にベンチマーク結果を示しますが、利用環境や PDF ファイルによって変動しますので参考程度とお考えください。

○ コマンドラインによる署名試験

コマンドラインを使い署名をループして繰り返し（1000 または 100 ループ）実行

実行マシン：Intel Core i7 3.4GHz/4core/mem 8GB/Windows 7 Pro x64

実行環境：CentOS 5.7 64bit/2CPU/mem 2.8GB

・実行マシン上の VMware で実行環境を利用

・タイムスタンプは付与していないので注意

入力ファイル	121 ページ 200KB	140 ページ 1MB	873 ページ 12MB	1,310 ページ 32MB
同時 1 実行(2core)	14.5 署名/秒	16.9 署名/秒	1.42 署名/秒	0.94 署名/秒
同時 2 実行(2core)	13.5 署名/秒	15.9 署名/秒	---	---
同時 4 実行(2core)	6.14 署名/秒	6.45 署名/秒	0.99 署名/秒	0.71 署名/秒

○ JAVA API による検証試験

JAVA API を使い検証をマルチスレッドで 64 同時実行

実行マシン：P4/Xeon 3GHz/4core/2CPU/mem 1GB/Linux Debian 5 32bit/RAID

・CRL は社内ネットなので高速の為ほぼ遅延無し

入力ファイル	3 ページ　44KB
同時 64 実行(4core×2CPU)	52.6 検証/秒

※ メモリ総利用量は 25～30MB 程度（Java 実行環境を含む）

※ メモリ利用量は PDF ファイルサイズに依存する（現在はメモリ上に保持の為）

Q	マルチスレッドに対応していますか？
A	はい対応しています。スレッド毎にインスタンスを生成して利用してください。

付録Ｄ．コピーライト表示

```
[OpenSSL License]
Copyright 2002-2020 The OpenSSL Project

Licensed under the Apache License, Version 2.0 (the “License”);
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is
distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and limitations under the License.

[libjpeg License]
This product includes softwareIeveloped by:
 Copyright (C) 1994-2013, Thomas G. Lane, Guido Vollbeding.
 This software is based in part on the work of the Independent JPEG Group
 http://www.ijg.org/

[libpng License]
This product includes softwIs developed by:
 Copyright (c) 2004, 2006-2012 GIn Randers-Pehrson.
 Copyright (c) 1998, 1999, 2000-2002 Glenn Randers-Pehrson
 Copyright (c) 1996, 1997 Andreas Dilger
 Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.
 http://www.libpng.org/pub/png/libpng.html

[OpenLDAP License]
This product includes softwares developed by:
 Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA.  All Rights Reserved.
 http://www.openldap.org/

[libxml2 License]
This product includes softwares developed by:
 Copyright (C) 1998-2003 Daniel Veillard.  All Rights Reserved.
 http://www.xmlsoft.org/

[iconv License]
This product includes softwares developed by:
 Copyright (C) 1999-2003 Free Software Foundation, Inc.
 http://www.gnu.org/licenses/lgpl.html

[zlib License]
This product includes softwares developed by:
 Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler.
 http://www.zlib.net/

[AES License]
This product includes softwares developed by:
 Copyright (c) 1998-2010, Brian Gladman, Worcester, UK. All rights reserved.
 http://www.gladman.me.uk/

[MD5 License]
This product includes softwares developed by:
 Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
 RSA Data Security, Inc. MD5 Message-Digest Algorithm
```

```
 derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm
 http://www.ietf.org/rfc/rfc1321.txt

[BigInt License]
Copyright (c) 2020 William Chanrico
https://github.com/williamchanrico/biginteger-cpp/blob/master/LICENSE
```

※ license フォルダ下に詳細なコピーライトファイルあり。

◆ 製品についてのお問合せ窓口
support@langedge.jp / TEL 03-3862-2268 / 担当：宮地

有限会社ラング・エッジ

〒101-0032 東京都千代田区岩本町２－１４－１２ 宮崎ビル２Ｆ
TEL 03-3862-2268 web https://www.langedge.jp/

以上