ラング・エッジ PKI 基本ライブラリ

LE:PKI:Lib マニュアル

２０２６年７月１日版

□ 目次

□ 目次 ……1
○ 履歴 ……3
１．概要……4
　１．１．API 環境 ……6
　１．２．動作環境 ……7
　１．３．動作に必要なファイル ……8
　１．４．Windows 環境のインストールとソースビルド ……9
　１．５．Linux 環境のインストールとソースビルド ……11
　１．６．内部プロジェクト構成 ……14
　１．７．利用外部ライブラリ ……15
　１．８．フォルダ構成 ……18
　１．９．文字コードと LeString クラス ……19
２．ＡＰＩ利用方法 ……20
　２．１．API リファレンス ……20
　２．２．Windows C++ 利用の場合……23
　２．３．Linux C++ 利用の場合 ……24
　２．４．Java API 利用の場合 ……25
　２．５．.NET API 利用の場合 ……27
　２．６．LpkCmd コマンドの利用例 ……29
３．ＰＫＩ要素と署名方法 ……30
　３．１．証明書と署名鍵（秘密鍵） ……30
　３．２．検証情報（証明書チェーン・CRL・OCSP）と検証手順 ……33
　３．３．タイムスタンプの利用 ……35
　３．４．独自証明書ストア（Linux 版/Windows 版共通） ……38
　３．５．Windows 証明書ストア（Windows 版のみ対応） ……39
　３．６．CAPI（CryptoAPI）対応 IC カードによる署名の利用 ……43
　３．７．PKCS#12 ファイルによる署名の利用 ……47
　３．８．PKCS#11 対応 IC カードによる署名の利用 ……48
　３．９．証明書検証サーバー（CVS）の利用：GPKI/LGPKI 等……49
　３．１０．LE:PKI:Lib 試験用環境（証明書やタイムスタンプ等） ……51
　３．１１．失効情報取得の高度な指定 ……54
　３．１２．CRL キャッシュ機能 ……57
　３．１３．ECDSA（楕円曲線暗号）対応……57
　３．１４．CNG（Cryptography API: Next Generation）対応 IC カードによる署名の利用 ……58
４．ネットワーク機能 ……61
　４．１．Windows 版：通信方式の指定 ……62
　４．２．HTTP 通信のプロキシ設定 ……63
付録Ａ．エラーコード ……68

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

Ａ．１．LeUtil エラーコード（LeUtil.h）-1000 ～ -1099 ……68
Ａ．２．LpkCmd エラーコード（LpkCmd.h）-1200 ～ -1399 ……68
Ａ．３．LePKI エラーコード（LePKI.h）-3000 ～ -3999 ……68
付録Ｂ．コピーライト表示 ……71

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

○ 履歴

バージョン	日付	主な修正内容
Ver 1.10.R1	2026/07/01	「3.6. CAPI（CryptoAPI）対応 IC カードによる署名の利用」に商業登記リモート署名のスペックを追加。 CNG の IC カードやリモート署名ドライバへの対応追加 「3.14. CNG 対応 IC カード」 Windows 開発環境に Visual Studio 2026 を追加して、2015 を削除。
Ver 1.09.R3	2025/08/27	Windows 版のプロキシ設定に関する更新（設定等の情報追加)。 「4.2. HTTP 通信のプロキシ設定」
Ver 1.09.R2	2025/01/27	Windows 版のプロキシ設定に関する更新。 「4.2. HTTP 通信のプロキシ設定」
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。 「3.13. ECDSA（楕円曲線暗号）署名」の追加
Ver 1.08.R3	2024/01/22	「4.1. Windows 版：通信方式の指定」の「２）WinHTTP 利用時のプロキシ設定（Windows 版のみ)」に認証方式の記述を追加。
Ver 1.08.R1	2023/08/30	LE:PAdES:Lib のマニュアルから LE:PKI:Lib の説明を独立し本書に統合 これに伴いこれまでの LE:PAdES:Lib での更新内容を反映した
Ver 1.06.R2	2019/02/01	2.1. API リファレンスの LpkCmd のヘルプを更新 タイムスタンプ検証時のオプションの追加
Ver 1.06.B2	2019/01/20	LE:PKI:Lib マニュアルの第１版

- 本製品マニュアルに記載の会社名、製品名は、各社の商標または登録商標です。
- 本製品マニュアルに記載の内容の一部または全部を無断で複写・転載することを禁じます。
- 本製品マニュアルに記載の内容及び製品の仕様等は予告なく変更される場合があります。最新情報はラング・エッジの製品ページで確認できます。
- 製品ページ　http://www.langedge.jp/biz/

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．概要

PKI とは公開鍵基盤の略称である。PKI 基本ライブラリ LE:PKI:Lib は、公開鍵基盤（PKI）に関連する署名や検証および各種情報の取得等を行う為の基本ライブラリである。PDF 長期署名ライブラリ LE:PAdES:Lib（LE:PAdES-Basic:Lib）や XML 長期署名ライブラリ LE:XAdE:Lib からも利用されている。Ver1.09（2024 年 2 月リリース）より ECDSA 署名に対応した。

○ Ver1.09.R2 の仕様変更に関する注意点

LE:PKI:Lib の Ver1.09.R2（2025 年 1 月リリース）では Windows 環境の HTTP 通信において、proxy.ini 設定ファイルの指定が可能となった。詳しくは LE:PKI:Lib マニュアル（LePKI-manual.pdf）の「４．２．HTTP 通信のプロキシ設定」の章を参照。

○ 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	３．１１．失効情報取得の高度な指定

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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	４．１．Windows 版：通信方式の指定

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１． １．API 環境

LE:PKI:Lib は C++で開発されており、インターフェイスとして C++ / Java / .NET（Windows 版のみ）/ コマンドの３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	Java8(1.8)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)	LpkCmd 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/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（Windows 版のみ）

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 LpkCmd 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。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．２．動作環境

LE:PKI: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 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++ 2017 と 2022 でビルド（C++ランタイムが必要） → リビルドすることで VS2019 / VS2026 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）

動作環境

リリース種別	種別	OS 環境	動作環境
LE:PKI:Lib Windows 版	LE:PKI:Lib	Windows	32bit/64bit 同梱
LE:PKI:Lib Linux 64bit 版		Linux	64bit ※
※ 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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．３．動作に必要なファイル

LE:PKI:Lib の動作に必要なファイルは大きく分けて LePKI 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
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 Java 用の LePKI クラスパッケージとモジュール
lepki-1.0.X.jar [java/lib]の下		
LePKIdnet.dll	（未サポート）	○ .NET 環境利用時のみ必要なモジュール
LpkCmd.exe	LpkCmd LpkCmd.sh [bin_linux]の下	○ 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

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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．４．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） LePKI-1.XX.RX.zip を Unzip 展開する。

1-1） 展開したフォルダ（LePKI-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 LpkCmd.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:PKI:Lib ファイル：
LePKI.dll、LePKIjni.dll、LePKIdnet.dll、LpkCmd.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、LpkCmd.exe が実行できることを確認する。以下は LE:PKI:Lib の例。

```
> LpkCmd.exe
LpkCmd : LePKI(V1.0X.RX) Langedge PKI Command.
CopyIht © 2012-202X LangEdge,Inc. all rights reserved.
>
```

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

2） Windows 環境のソースビルド

2-1） LE:PKI:Lib の src フォルダ中にある LePKI2015.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 “LpkCmd” 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¥LePKI2013.sln
Visual Studio 2015	src¥LePKI2015.sln
Visual Studio 2017	src¥LePKI2017.sln **※ 製品リリースのビルドに利用**
Visual Studio 2019	src¥LePKI2019.sln
Visual Studio 2022	src¥LePKI2022.sln **※ 製品リリースのビルドに利用**
Visual Studio 2026	src¥LePKI2026.sln

※ Ver1.08 より Visuasl Studio 2010 / 2012 は非サポートとなった。
※ Ver1.10 より Visuasl Studio 2013 / 2015 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。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．５．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） LePKI-1.XX.RX.tar.gz を tar 展開する。

1-1） 展開したフォルダ（LePKI-1.XX.RX）下にある bin_linux/LpkCmd.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：$ LePKI-1.XX.RX/bin_linux/LpkCmd.sh
LpkCmd : LePKI(V1.XX.RX) LangEdge PKI Command.
CIright (c) 2012-202X LangEdge,Inc. all rights reserved.
（以下略）

1-2） LpkCmd/C++/Java 各 API の動作基本テストを行い、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。

例：$ make test

※ コマンド（LpkCmd.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。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

○ 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（LePKI-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/LePKI-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（LePKI-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/lib64 の下であれば make –f Makefile.pki install や make –f Makefile.pki install32 でも良い。

例 1(64bit 環境)：

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

例 2(32bit 環境)：

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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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（LePKI-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
```

※：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） 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:PKI:Lib のルートディレクトリ直下にある Makefile.pki 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 pki でも同じ
$ make all32	一括ビルド(32bit 版)
$ make test	テスト(sample)実行 ※ $ make test.pki でも同じ
# make install	/usr/lib64 へインストール実行(64bit 版) ※ルート権限が必要
# make install32	/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。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．６．内部プロジェクト構成

内部プロジェクトは PDF 操作の LePKI 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
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	
	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
LpkCmd.exe (LpkCmd)	LpkCmd	PKI 操作のコマンド	
	[LePKI]	PKI 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	---
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:PKI:Lib の内部プロジェクト構成（src フォルダの下にある）

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．７．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
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

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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

※ 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.
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.

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:PKI: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.

 [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]
```

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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

コピーライト表示

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．８．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。	LpkCmd 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。 [LePKI][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 / LePKI / LePKI に分かれている。	C++ API 利用時必要
[doc]	C++/Java/.NET の API ドキュメント他のフォルダ。 [LePKI-manual.pdf] 製品マニュアル [LePKI/cpp/index.html] C++ API リファレンス [LePKI/java/index.html] Java API リファレンス [LePKI/dotnet/index.html] .NET API リファレンス [LePKI/cmd/LpkCmd-Help.txt] LpkCmd ヘルプ	リファレンス用
readme-LePKI.txt	LE:PKI:Lib に関する説明ファイル。	お読みください
[sample]	[LePKI/cmd] LpkCmd（コマンド）利用サンプル [LePKI/cpp] C++利用サンプル [LePKI/cs] C# 利用サンプル [LePKI/java] java 利用サンプル	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（LePKI/LePKI 等）用のフォルダ [src/LePKI2019.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
[license]	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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

１．９．文字コードと LeString クラス

LE:PKI: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。

LpkCmd 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 LpkCmd 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。

LpkCmd 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。

LpkCmd の出力メッセージのオプション指定

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

２．ＡＰＩ利用方法

２．１．API リファレンス

以下に LE:PKI:Lib のクラス構成を示す。

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

```
LpkCmd : LePKI(V1.10.R1) LangEdge PKI Command.
         Copyright (c) 2012-2026 LangEdge,Inc. all rights reserved.

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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

```
command: 操作を指定 (必須)
  -sign    : CAdES/PKCS#7 生成または TimeStamp トークン取得
  -verify  : CAdES/PKCS#7/TimeStamp/証明書の署名値と証明書の検証

options: オプション指定
  -type [cades/pkcs7/ts/cert] : 署名種別の指定 (必須)
     cades : CAdES 長期署名形式 (CAdES-BES/CAdES-T)
     pkcs7 : PKCS#7 署名形式
     ts    : RFC3161 タイムスタンプ形式
     cert  : X.509 証明書形式 (-sign は非対応)

options: 署名オプション
  -cert <p12/x509/finger/select/card/p11> : 署名証明書指定 (署名時必須)
     p12 filepath passwd : PKCS#12 指定 P12 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 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 : CRPKI(CAPI) : 'CRPKI Crypto Service Provider for Remote Sign'
       8 : Pentio/TDB/LGPKI : MS_SCARD_PROV_A (MINI DRIVER)
     card cng [numb]     : CNG の IC カード利用 (種別指定)
       1 : CRPKI(CNG) : 'CRPKI Key Storage Provider for Remote Sign'
     p11 filepath passwd : PKCS#11 指定 P11 の DLL ファイルとパスワードが必要
  -ts <3161/amano> : タイムスタンプ指定 (ts 必須)
     ※ pkcs7/cades では署名タイムスタンプ
     3161 url [hash] [id] [passwd] : RFC3161 (id/passwd 指定で Basic 認証対応)
                                     hash には<sha1/s512>が指定可能
     amano url license_file passwd : AMANO タイムスタンプサービス(有償)
                                     ライセンスファイルとパスワードが必要
  -tshash    : タイムスタンプ対象を-target にハッシュ値 HEX 文字列で指定

options: 検証オプション
  -detail    : 詳細な検証情報を出力
  -tst <time/hash/cert> : タイムスタンプトークンの情報を出力
     time    : タイムスタンプ時刻を出力
     hash    : タイムスタンプ対象ハッシュ値を HEX 文字列出力
     cert    : TSA 証明書の情報を出力
  -tsa <filepath> : TSA 証明書をバイナリ出力する(タイムスタぷのみ)
  -tflag <all/none/sign|hash|cert> : タイムスタンプ検証フラグ
     all     : 全ての検証を行う(標準設定/-target 引数必須)
     none    : 検証しない(-tst オプションで情報のみ取得)
     sign    : トークンの TSA 証明書による署名を検証
     hash    : ハッシュ値を比較(-target 引数必須)
     cert    : TSA 証明書の PKI 検証
  -sflag <none/org|win> : 証明書ストアフラグ
     none    : 証明書ストア無し(試験用)
     org     : 独自証明書ストア利用
```

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

```
      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)
  -prior crl : 失効情報優先フラグ指定
     crl : CRL 優先、省略時は OCSP 優先
  -repository url  : ディレクトリサーバの指定
  -time GeneralizedTime : 検証時刻の指定 省略時=現在 (試験用)
  -http <http|inet|sock> : HTTP 通信 API 種別の指定 (省略時は http)
     http : WinHTTP 利用 (推奨/省略時設定)
     inet : WinInet 利用 (非推奨/過去互換)
     sock : 独自 Socket 利用 (OpenSSL 利用)

options: 情報オプション
  -hash <sha1/s256/s384/s512> : ハッシュ値の表示 (-type オプション不要)
     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)
  -base64 : ハッシュ値を Base64 表示 (デフォルト : Hex)

options: 入出力オプション
  -target <filepath>  : 署名/タイムスタンプの対象ファイル (必須)
  -target <hex>       : -tshash 指定時には HEX 文字列でハッシュ値を指定
  -in <filepath>      : 入力ファイル (verify コマンド必須)
  -out <filepath>     : 出力ファイル (sign コマンド必須)

options: ヘルプオプション
  -help               : ヘルプ表示
  -eng / -sjis / -utf8 : メッセージを英語(eng)又は指定文字コードで表示

例) CAdES 署名生成
 > LpkCmd -sign -type cades -cert p12 cert.p12 PASSWORD ¥
          -target test.txt -out test.p7s

例) CAdES 署名検証
 > LpkCmd -verify -type cades -target test.txt -in test.p7s

例) タイムスタンプ取得
 > LpkCmd -sign -type ts -ts 3161 http://www.langedge.jp/tsa s512 ¥
          -target test.txt -out test.tst

例) タイムスタンプ検証
 > LpkCmd -verify -type ts -target test.txt -in test.tst

例) 証明書検証
 > LpkCmd -verify -type cert -detail -in test.cer
```

LpkCmd-Help.txt

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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（LePKI-1.XX.RX）下にある include ディレクトリ。
LePKI.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:PKI:Lib インクルードファイル
#include <LePKI/LePKI.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

LE:PKI: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:PKI:Lib インターフェイスライブラリファイル
#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/LePKI/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
CppVerifyTest.bat CppVerify.cpp	検証サンプル

Windows C++用のサンプル

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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（LePKI-1.XX.RX or LePKI-Basic-1.XX.RX）下にある include ディレクトリ。

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

LePKI.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:PKI:Lib インクルードファイル
#include <LePKI/LePKI.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（LePKI-1.XX.RX or LePKI-Basic-1-XX.RX）下にある lib_linux ディレクトリ。

リンク時の引数に -lLePKI により libLePKI.so を指定

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

※ 実例として sample/LePKI/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
CppSignTest.sh CppSign.cpp	署名サンプル
CppVerifyTest.sh CppVerify.cpp	検証サンプル

Linux C++用のサンプル

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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（LePKI-1.XX.RX or LePKI-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 LePKI と LePKI を以下のようにインポートしておく。

```
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 LePKI-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 .;lepki-1.0.XX.jar Sample.java
// 実行
java -classpath .;lepki-1.0.XX.jar Sample
```

※ 実例として sample/LePKI/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
JavaSignTest.bat JavaSignTest.sh LePKI_sign.java	署名サンプル
JavaVerifyTest.bat JavaVerifyTest.sh LePKI_verify.java	検証サンプル

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:PKI: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 LePKI である。LE:PKI: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。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。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

クラス名	ネイティブメモリ利用	補足
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

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:PKI: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 LePKI-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/LePKI/JavaBuild.bat (JavaBuild.sh)

OpenJDK 系での利用も問題ありません。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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 LePKIdnet.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	その他 DLL (アンマネージド DLL)
LePKI	LePKIdnet.dll	LePKI.dll	libeay32L.dll, ssleay32L.dll, libxml2.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="LePKIdnet" /> </dependentAssembly> <developmentMode developerInstallation="true" /> </runtime> </configuration>	Configuration 開始 runtime 指定 dependentAssembly 設定 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：LePKIdnet.dll を参照に追加する

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

ファイル	概要
CsBuild2005.bat , CsBuild2010.bat	バッチ式ビルド一括実行
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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

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:PKI: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 LePKI と LePKI である。

LE:PKI: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。LePKI 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
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

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:PKI: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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

２．６．LpkCmd コマンドの利用例

LE:PKI: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 LpkCmd が提供されている。利用方法の詳細はヘルプンド（LpkCmd -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/タイムスタンプ取得

○ CAdES 署名付与（PKCS#12 ファイルの利用・タイムスタンプ有り）
（%TS_URL%で URL 指定した例）

```
＞ LpkCmd.exe -sign -type cades -cert p12 ../LeTest.p12 test ¥
   -ts 3161 %TS_URL% -target target.txt -out cades-t.p7s
```

○ PKCS#7 署名付与（タイムスタンプ無し）

```
＞ LpkCmd.exe -sign -type pkcs7 -cert p12 ../LeTest.p12 test -target target.txt ¥
   -out pkcs7.p7s
```

○ RFC 3161 タイムスタンプ取得
（%TS_URL%で URL 指定した例）

```
＞ LpkCmd.exe -sign -type ts -ts 3161 %TS_URL% -target target.txt -out timestamp.tst
```

３）署名/タイムスタンプ検証

○ CAdES 署名検証

```
＞ LpkCmd.exe -verify -type cades -detail -store ../store/ -target target.txt -in cades-t.p7s
```

○ PKCS#7 署名検証

```
＞ LpkCmd.exe -verify -type pkcs7 -detail -store ../store/ -target target.txt -in pkcs7.p7s
```

○ RFC 3161 タイムスタンプ検証

```
＞ LpkCmd.exe -verify -type ts -detail -store ../store/ -target target -in timestamp.tst
```

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

ファイル	概要
CmdSignTest.bat CmdSignTest.sh	署名サンプル（署名フィールドも同時に作成）
CmdVerifyTest.bat CmdVerifyTest.sh	検証サンプル

コマンド利用のサンプル

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．ＰＫＩ要素と署名方法

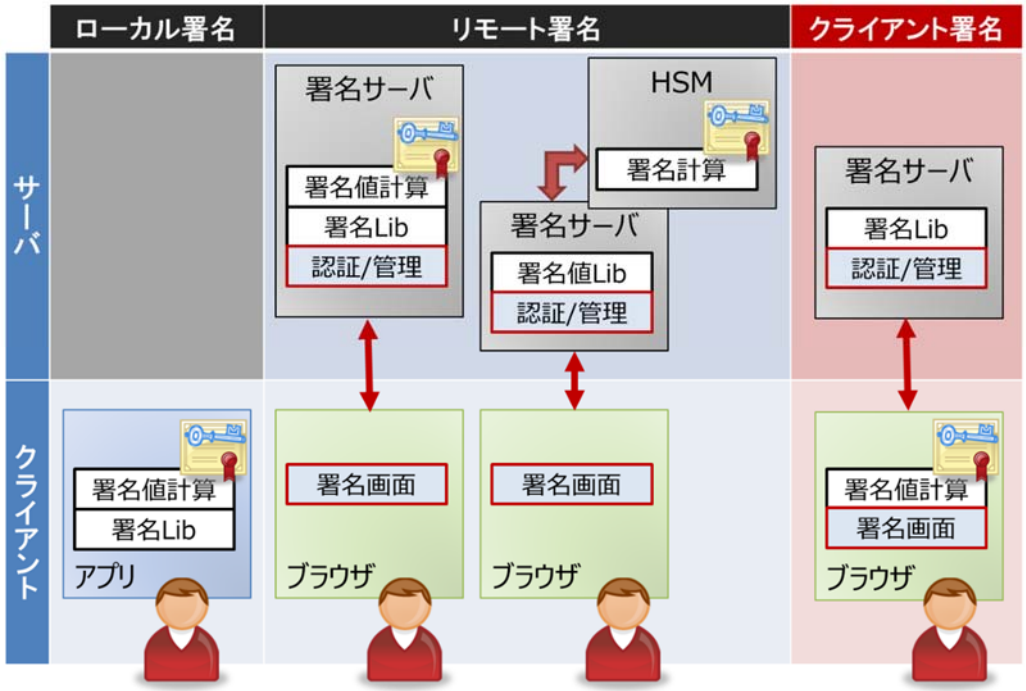
３．１．証明書と署名鍵（秘密鍵）

LE:PKI:Lib において、証明書は X.509 標準の電子証明書をサポートしている。X.509 電子証明書は LpkCert クラスを利用する事で利用が可能になっている。証明書には公開鍵が含まれているが、ペアとなる署名鍵（秘密鍵とも呼ばれる）が署名時に必要となる。署名鍵はソフトウェア（ファイルやデータ）として保持する方法と、ハードウェア（HSM や IC カード等）として保持する方法の 2 種類がある。また署名鍵をサーバーで管理するのか、利用者の PC 上で管理するのか、と言う点も署名システムを設計する上では重要となる。電子署名の実現方式は大きく分けると「ローカル署名」「リモート署名」「クライアント署名」の 3 つに分類できる。

1. **ローカル署名（署名アプリ等）**
   - ➢ クライアント端末上で署名処理**全て**を行う
     - ✓ タイムスタンプ取得や検証時にはネット接続が必要

2. **リモート署名（クラウド署名）**
   - ➢ サーバー上で署名処理**全て**を行う
     - ✓ 利用者の認証が必須であり近年注目を集める方式
     - ※ 参考：JT2A リモート署名ガイドライン（JNSA サイト）
       https://www.jnsa.org/result/jt2a/data/RemoteSignatureGguide_All.pdf

3. **クライアント署名（サーバー連携）**
   - ➢ サーバーとクライアントが**連携**して署名を行う（ラング・エッジの造語）
     - ✓ サーバー部は署名ファイルの生成とハッシュ値計算
     - ✓ クライアント部はハッシュ値から署名値の生成のみ



電子署名の実現方法の分類

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

ローカル署名とリモート署名は署名鍵と署名ライブラリである LE:PAdES:Lib や LE:XAdES:Lib を同じ PC・システム上で利用し、証明書の LpkCert クラスを使って署名することになる。ただし HSM を利用する場合は署名値の計算を外部にある HSM で計算する為に LpkCert クラスは使わずに独自に署名値計算部を実装する必要がある。クライアント署名は利用者 PC（Windows 環境）上のブラウザで動作する LE:Client:Sign が提供する専用プラグインで署名値を計算する。

種類	Linux	Windows	概要と詳細章
Windows 証明書ストア	×不可	**〇可能**	４．４．Windows 証明書ストア を参照
PKCS#12 ファイル	**〇可能**	**〇可能**	４．５．PKCS#12 ファイル を参照
CAPI 対応 IC カード （IC カードリーダーが必要）	×不可	**〇可能**	V1.07.R2 以降 CAPI スペック指定で利用可能 ４．６．CAPI 対応 IC カード を参照 ※ V1.07.R1 以前は JPKI のみ利用可能
PKCS#11 対応 IC カード （IC カードリーダーが必要）	△限定	△限定	機能はあるが実績が少なく注意が必要。 ４．７．PKCS#11 対応 IC カード を参照
HSM 等署名値計算は別 （HSM は別サーバーで稼働）	**〇可能**	**〇可能**	４．８．HSM を参照 ※ 署名に LpkCert クラスを使わない利用
クライアント署名 （Windows ブラウザ用プラグイン）	**〇可能** (サーバー側)	**〇可能** (サーバー側)	サーバー側の JavaScript ライブラリと利用者 PC 上のブラウザ（IE か Edge）のプラグインと連携して PAdES/ XAdES 署名を実現。署名値のみ利用者 PC で計算する。 ７．クライアント署名（Ver2.1） を参照 ※ 署名値計算には専用プラグインを利用

署名鍵の保管種類による署名方法の分類

証明書は認証局にて発行されるが、日本において公的に認められている証明書は大きく分けると以下の５種類がある。このうち最初の２つ（官職証明書とヘルスケア PKI）は一般では利用できない。法人であれば商業登記証明書が使えるが、大企業では難しい為に特定認証局を利用するケースが多い。個人であれば公的個人認証サービスの利用が簡単である。これらの多くは PKCS#12 形式で提供されるが IC カード等で提供されるケースもある。IC カードは CAPI 対応または PKCS#11 対応としてドライバが提供されるが全てに対応出来る訳では無い。IC カードの利用について詳しくは「４．６ CAPI 対応 IC カードによる署名の利用」と「４．７ PKCS#11 対応 IC カードによる署名の利用」を参照。

証明書	発行	説明
官職証明書	官職認証局	GPKI やほぼ同じ仕様の LGPKI 等
ヘルスケア PKI	厚生労働省	医療従事者向けの HPKI
商業登記(電子認証登記所電子証明書)	法務省	法人代表者に発行される証明書
特定認証業務(一般的に特定認証局)	民間認証局	GPKI で認められ相互接続された認定証明書
公的個人認証サービス電子証明書	地方自治体	公的個人 JPKI 証明書（マイナンバーカード）

日本で公的に認められる証明書

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

世界標準の認証局として WebTrust 認定され Windows（IE/Edge）やブラウザに登録済みの認証局も多い。これらの証明書も PKCS#12 形式で入手可能であれば、LE:PKI:Lib にて利用可能である。WebTrust 認定された証明書はルート証明書をインストールする手間も無く使いやすいが、公的な申請等では利用できないので注意が必要。証明書の選択は別途コンサルも可能。

なお証明書の失効については一般的に、CRL（証明書失効リスト）による方法と、OCSP（オンライン証明書状態プロトコル）による方法の、いずれかとなる。LE:PKI:Lib ではどちらにも対応している。また失効情報（CRL/OCSP）には署名が付けられるが、証明書の署名と同じ署名証明書では無い場合でも信頼済みのルート証明書であれば有効となる。

PDF 署名では業界標準となっている Adobe Reader/Acrobat では、失効情報への署名も証明書の署名と同じ署名証明書である必要があるのでこの点では検証結果が異なる可能性がある。これは対応しているポリシの違いである。他にも証明書検証において既存の検証器と検証結果が異なるケースもあるがいずれも対応している仕様やポリシの違いであり異常では無い。

参考情報：

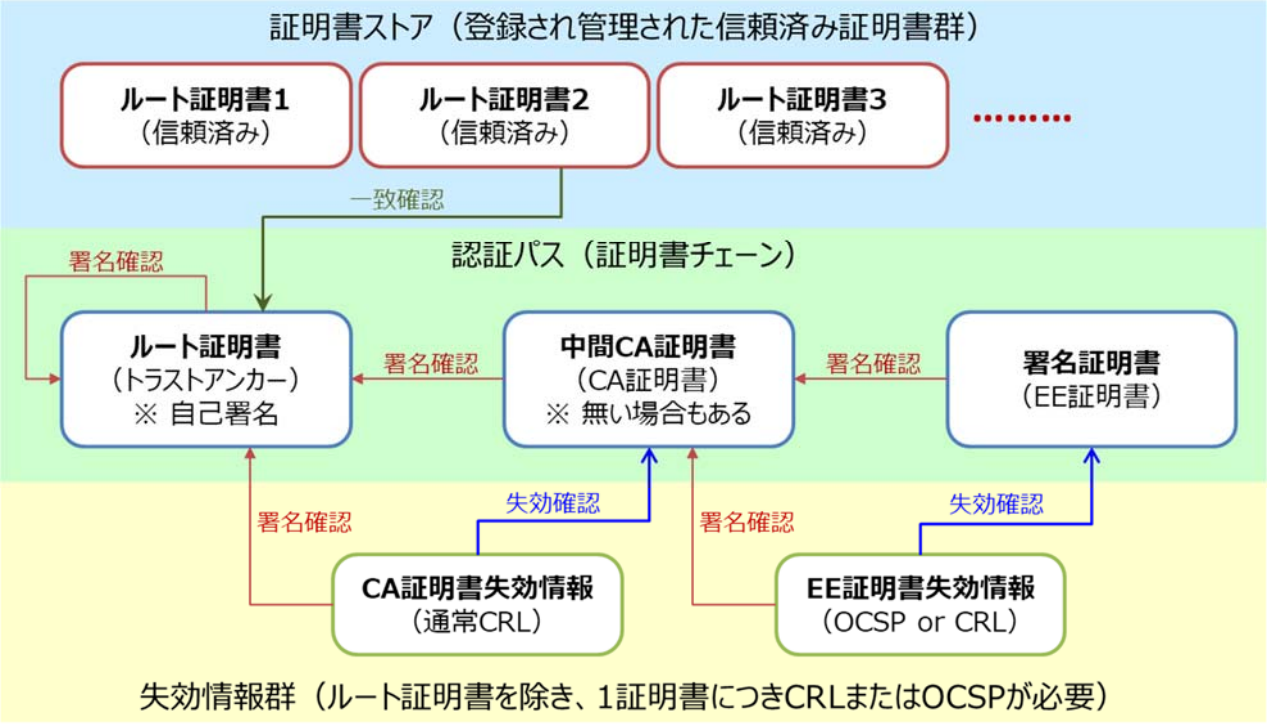
デジタル署名検証ガイドライン（JNSA 電子署名ワーキンググループ）
https://www.jnsa.org/result/e-signature/2021/

電子署名検証を 10 分で説明してみる
http://eswg.jnsa.org/matsuri/201610/20161026-L4-miyachi.pdf

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．２．検証情報（証明書チェーン・CRL・OCSP）と検証手順

X.509 電子証明書は PKI（公開鍵基盤）により検証が可能となっている。検証時に必要な情報は検証情報と呼ばれる。検証情報は認証パスに含まれる証明書と CRL や OCSP により構成される。署名証明書（EE：エンドエンティティ証明書）は発行元の親証明書（以下例だと中間 CA 証明書）から順に自己署名のルート証明書までの認証パスを順に辿って構築する（以下例だとルート証明書・中間 CA 証明書・署名証明書の 3 階層）。認証パスのルート証明書を除く各証明書に対して失効情報により失効確認を行う。PKI の起点となるルート証明書（自己署名証明書）の失効確認は必要なく証明書ストアの「信頼済み」証明書群にあることを確認する。



3 階層の認証パスを持つ場合の検証情報関連図の例

LE:PKI:Lib では証明書を検証する API として LePKI.verifyCert() を提供している。verifyCert() は内部では、認証パスを構築する buildPath() と、失効情報を取得する validPath() と、失効有無の確認とトラストアンカーの確認を行い検証する verifyPath() を実行している。

目的	LePKI メソッド	説明
証明書検証	verifyCert	内部で buildPath → validPath → verifyPath を実行。
認証パス構築	buildPath	指定証明書から自己署名証明書までの認証パスを構築する。
失効情報取得	validPath	認証パスの各証明書の失効情報を取得する。
失効情報等確認	verifyPath	各証明書が有効でありルート証明書が信頼済みを確認する。

LePKI クラスの証明書検証 API

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

LePKI.validPath では色々な場所から検証情報を取得する。各証明書/CRL/OCSP はどこから取得したものかを LpkCert/LpkCrl/LpkOcsp 各クラスの getFrom() で確認できる。長期署名済みの場合には PAdES/XAdES の検証情報から取得されるが、長期署名ではない場合には証明書ストアやネットワークから取得される。長期署名と呼ぶ為には最後のアーカイブタイムスタンプ関連の検証情報を除き LPK_FRM_NETWORK が無い必要がある。

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	CAdES/PKICS#7 等の署名データから取得
LPK_FM_ORGSTORE	orgStore	ラング・エッジの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（Windows 版のみ）
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）から取得した証明書
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（Ver1.08 では未サポート）
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

失効有無の確認は失効リストの CRL（ブラックリス方式：リストに含まれなければ有効）と有効性確認の OCSP（証明書毎に問合せして失効有無を返却）のいずれかで行われる。証明書によっては CRL と OCSP の両方に対応している場合もあるがどちらを使うかは「３．１１．失効情報取得の高度な指定」を参考に決める必要がある。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．３．タイムスタンプの利用

LE:PKI:Lib において、タイムスタンプは RFC 3161 標準およびアマノタイムスタンプサービス 3161 のライセンスファイルをサポートしている。RFC 3161 標準を利用する場合は LpkTs3161 クラスを、アマノタイムスタンプサービス 3161 のライセンスファイルを利用する場合は LpkTsAmano クラスを使う。日本において公的に認められているタイムスタンプ局は、総務大臣認定を受ける必要がある。認定されたタイムスタンプ局は総務省のページで確認が出来る。認定を受けたタイムスタンプ局を本書では「商用タイムスタンプ局」と呼ぶ。商用タイムスタンプ局の利用は通常有償である。

総務省：総務大臣による（タイムスタンプ）認定制度
https://www.soumu.go.jp/main_sosiki/joho_tsusin/top/ninshou-law/timestamp.html

ラング・エッジでは試験用にオープンなタイムスタンプサーバーを用意している。これは OpenSSL を利用した FreeTSA を利用している（「３．１０．LE:PKI:Lib 試験用環境」参照）。ラング・エッジの電子署名製品の利用サンプルは全てこの FreeTSA を使った試験用タイムスタンプサーバーを利用している。なお独自の TSA 証明書を利用しているので公式や公的な目的では利用できない。あくまで動作試験用として運用しているので連続した過度なアクセスは認められない。

目的	LE:PKI:Lib クラス	説明
タイムスタンプ取得のベースクラス	LpkTimestamp	C++においてタイムスタンプ取得の API を示すベースクラス。LpkTs3161 と LpkTsAmano が参照している。Java と .NET では直接 LpkTs3161 と LpkTsAmano クラスを使う為に利用する必要は無い。
標準の RFC 3161 利用したタイムスタンプ取得	LpkTs3161	RFC 3161 標準に準拠した HTTP/HTTPS 通信によりタイムスタンプ取得する為のクラス。Basic 認証に対応している。
アマノのサービスのタイムスタンプ取得	LpkTsAmano	アマノタイムスタンプサービス 3161 のライセンスファイルを利用してタイムスタンプ取得する為のクラス。
タイムスタンプトークン利用	LpkTimestampToken	LpkTs3161 と LpkTsAmano で取得したタイムスタンプトークンを利用する為のクラス。

LE:PKI:Lib のタイムスタンプ関連クラス

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

次に LE:PKI:Lib にてタイムスタンプを利用する場合の組み込み方法を簡単に説明する。

A）RFC3161 標準による接続

API からは専用クラス LpkTs3161 クラスを利用する。コマンドからは-ts 3161 引数で指定。

C++ API	le::LpkTs3161 ts; int rc = ts.set("https://サーバーURL", le::LPK_SHA_512); if(rc < 0) goto ERROR;
Java API	LpkTs3161 ts = new LpkTs3161(); Int rc = ts.set("https://サーバーURL", LePKI.LPK_SHA_512); if(rc < 0) return ERROR;
.NET API	LpkTs3161 ts = new LpkTs3161(); Int rc = ts.set("https://サーバーURL", LPK_HASH.LPK_SHA_512); if(rc < 0) return ERROR;
LpkCmd, LpaCmd, Lx3Cmd	LpkCmd -sign … -ts 3161 "http://サーバーURL" s512 …

※ LpkTs3161 クラスで指定可能なハッシュ方式は LPK_SHA_256 か LPK_SHA_512 です。

※ 試験用 FreeTSA の場合にはサーバーURL は https://www.langedge.jp/demotsa となります。

○ Basic 認証による接続（オプション）

C++ API	le::LpkTs3161 ts; int rc = ts.set("https://サーバーURL", le::LPK_SHA_512); if(rc < 0) goto ERROR; **rc = ts.basic("ID", "PASSWORD");** **if(rc < 0) goto ERROR;**
Java API	LpkTs3161 ts = new LpkTs3161(); Int rc = ts.set("https://サーバーURL", LePKI.LPK_SHA_512); if(rc < 0) return ERROR; **rc = ts.basic("ID", "PASSWORD");** **if(rc < 0) return ERROR;**
.NET API	LpkTs3161 ts = new LpkTs3161(); Int rc = ts.set("https://サーバーURL", LPK_HASH.LPK_SHA_512); if(rc < 0) return ERROR; **rc = ts.basic("ID", "PASSWORD");** **if(rc < 0) return ERROR;**
LpkCmd, LpaCmd, Lx3Cmd	LpkCmd -sign … -ts 3161 "http://サーバーURL" s512 **ID PSWD** …

※ 商用タイムスタンプの場合サーバーURL と ID/パスワードは別途ベンダーから提供されます。

※ 試験用 FreeTSA の場合にはサーバーURL は https://www.langedge.jp/batsa となり、ID が "langedge"でパスワードが"test"となります。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

B）アマノタイムスタンプサービス 3161 の利用（アマノのライセンスファイル利用）

API からは専用クラス LpkTsAmano クラスを利用する、コマンドからは-ts amano 引数で指定。

C++ API	le::LpkTsAmano ts; **int rc = ts.set("https://サーバーURL", "ライセンスファイル", "PASSWORD");** **if(rc < 0) goto ERROR;**
Java API .NET API	LpkTsAmano ts = new LpkTsAmano(); **Int rc = ts.set("https://サーバーURL", "ライセンスファイル", "PASSWORD");** **if(rc < 0) return ERROR;**
LpkCmd, LpaCmd, Lx3Cmd	LpkCmd -sign … -ts amano "http://サーバーURL" **LIC PSWD** …

※ サーバーURL とライセンスファイル、パスワードはアマノとの契約後に提供されます。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．４．独自証明書ストア（Linux 版/Windows 版共通）

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:PKI: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:PKI: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」の利用も可能となっており独自証明書ストアと共に利用することができる。

「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 LpkCmd コマンドであれば、LpkCmd の下にあるディレクトリとなる。サーバー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（と署名鍵）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

機能	LePKI メンバ	説明
独自証明書ストア設定	setStrore	ストアディレクトリのパスを指定する。
信頼済みルート証明書追加	addTrust	信頼済みルート証明書を追加する。※
中間 CA 証明書追加	addCert	中間 CA 証明書を追加する。※
失効リスト CRL 追加	addCrl	失効リスト CRL を追加する。※
失効確認 OCSP 追加	addOcsp	失効確認の OCSP レスポンスを追加する。※
※ 追加される情報の情報取得種別は LPK_FM_ADDSTORE となる。		

独自証明書ストアを利用する LePKI クラスの API

LE:PKI: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。Windows 版でも 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。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．５．Windows 証明書ストア（Windows 版のみ対応）

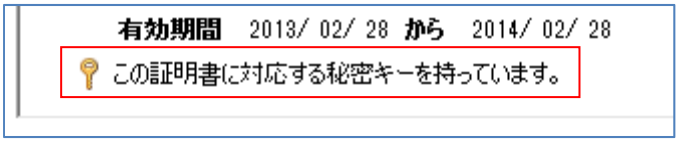
現在 Linux 版では独自証明書ストアのみの対応だが、Windows 版ではシステム標準の Windows 証明書ストアも署名や検証時に利用が可能である。Windows 証明書ストアは用途により 6 種類に分かれているが署名時と検証時に利用可能な Windows 証明書ストアは以下となる。なお検証時には Windows 証明書ストアの利用はオフにする事も出来る。

名称	指定名称	署名時	検証時	
個人	MY	○利用	×	署名証明書指定に利用
中間証明機関	CA	×	○利用	検証の中間証明書で利用 独自の certs フォルダと同じ
信頼されたルート証明機関	ROOT	×	○利用	検証の信頼点の確認に利用 独自の trusts フォルダと同じ
ほかの人	AddressBook	×	×	署名/検証には利用しない
…	…	×	×	他の名称のストアは利用しない

利用される Windows 証明書ストアの種類

検証時には中間証明書やトラストアンカー（信頼点）の確認に利用されるが、利用方法としては使う/使わない程度であるが、署名時の証明書指定は IC カード等も含み少し面倒な面がある。

Windows 証明書ストアの「個人」には通常インストール済みの署名鍵と連携した証明書が格納されている。Windows 環境で署名鍵と連携した証明書を表示した場合「全体」タブ中にて「この証明書に対応する秘密キーを持っています。」と表示される。



Windows 環境で署名鍵と連携した証明書を表示した場合

この署名鍵はログインしたユーザ毎に設定が異なる。ログインしたユーザであれば PIN やパスワード等の入力は不要であり、そのまま署名等に利用できる。サーバーの Web サービス等で利用する場合にはログインするユーザ名が重要となるので注意が必要である。

またインストールしていなくても、IC カードや USB トークンに格納された証明書と署名鍵も、Windows 証明書ストアの「個人」に表示される場合がある。なお IC カードの種類によっては別途 Windows 証明書ストアへの証明書インストールが必要な場合もあるので注意が必要である。

次に Windows 証明書ストアに個人に入っている証明書と関連付いた署名鍵の利用方法を指定方法別に簡単に説明する。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

1）証明書選択画面による証明書（＋署名鍵）の指定

Windows 標準の GUI を使った証明書と署名鍵の指定。GUI を使っている為にバッチ処理では利用できないので注意。title 引数で証明書選択画面に表示する文字列を指定可能。

API	利用方法
LpkCmd LpaCmd Lx3Cmd	-cert オプションの select 引数で指定 利用方法： -cert select 例）LpkCmd -sign -cert select …
C++	LePKI クラスの getCertSelect () メソッドで LpkCert 取得 利用方法： int getCertSelect(LpkCert& cert, void* hWnd, const CHAR* title);
Java	LePKI クラスの getCertSelect () メソッドで LpkCert 取得 利用方法： LpkCert getCertSelect(String title);

証明書選択画面による証明書と署名鍵の指定

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

2）ハッシュ値（又は証明書ファイル）による証明書（＋署名鍵）の指定

証明書のハッシュ値（指紋）を使った証明書と署名鍵の指定。X.509 形式の証明書ファイルを指定する事も可能。また C++のみだが Win32 の証明書ハンドル PCCERT_CONTEXT による指定も可能。

API	利用方法
LpkCmd LpaCmd Lx3Cmd	-cert オプションの finger 引数で指定 利用方法： -cert finger <hex> 例）LpkCmd -sign -cert finger 6DFABB6(略)38F7E72C …
C++	LePKI クラスの getCertSelect () メソッドで LpkCert 取得 利用方法： int getCertByHash(LpkCert& cert, const CHAR *hash, const CHAR* storeType); ※ storeType は通常 NULL（”MY” 指定と同じ）で良い。
Java	LePKI クラスの getCertSelect () メソッドで LpkCert 取得 利用方法： LpkCert getCertByHash (String hex, String storeType); ※ storeType は通常 null（”MY” 指定と同じ）で良い。

ハッシュ値による証明書と署名鍵の指定

API	利用方法
LpkCmd LpaCmd Lx3Cmd	-cert オプションの x509 引数で指定 利用方法： -cert x509 <path> 例）LpkCmd -sign -cert x509 cert/sign.cer …
C++	LpkCert クラスの setFile() / setBin() メソッドで指定、ファイルパスかバイナリ指定 利用方法： int setFile(const CHAR* path); / int setBin(const BINARY& bin);
Java	LpkCert クラスの setFile() / setBin() メソッドで指定、ファイルパスかバイナリ指定 利用方法： int setFile(String path); / int setBin(byte[] bin);

証明書ファイルによる証明書と署名鍵の指定

API	利用方法
C++	LpkCert クラスの setCapi() メソッドで指定、第 2 引数で IC カード用かどうか指定 利用方法： int setCapi(const void* pcCert, bool iccard); ※ 第 2 引数 iccard を true にすると GPKI 等 IC カード対応となる、通常 false を指定

PCCERT_CONTEXT による証明書と署名鍵の指定

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

※ Windows の CryptoAPI（CAPI）による RSA-SHA2 署名の考察

Windows 環境にて署名を行う場合に幾つかの API が提供されているが、LE:PKI:Lib では CAPI（CryptoAPI）を利用している。CAPI による暗号機能は暗号プロバイダ（CSP）を指定して行われる。標準の暗号プロバイダにおけるプロバイダタイプは PROV_RSA_FULL になる。IC カードや USB トークンの場合は暗号プロバイダが標準とは異なるが、通常プロバイダタイプは PROV_RSA_FULL の指定が必要となり別のプロバイダタイプは指定できない事が多い。

Windows 標準の暗号プロバイダではプロバイダタイプ PROV_RSA_FULL の場合に RSA-SHA1 署名しかサポートされていない。RSA-SHA2 署名に対応するにはプロバイダタイプ PROV_RSA_AES の指定が必要となる。一方 Windows 証明書ストアから証明書と署名鍵を取得するにはプロバイダタイプ PROV_RSA_FULL である必要がある。この為に署名前に署名鍵のプロバイダタイプを PROV_RSA_AES に変更する必要がある。

一方 IC カードの暗号プロバイダでは実装に依存する。GPKI の IC カードではプロバイダタイプ PROV_RSA_FULL のままで RSA-SHA2 署名も実行が可能となっている。

この為に同じ CAPI を使った RSA-SHA2 署名をする場合に、標準（Windows 証明書ストア）にある署名鍵を使う場合と、IC カードにある署名鍵を使う場合では処理が異なる。LpkCert::setCapi() の第 2 引数でカード利用の有無が指定できるのも、LpkCert::setCard() で CAPI 指定が可能なのも、この処理手順の違いとなる。

署名手順	RSA-SHA1	RSA-SHA2 ※1 証明書ストア	RSA-SHA2 ※2 IC カード等
PROV_RSA_FULL で CSP 初期化	○	○	○
署名する証明書の取得	○	○	○
証明書に関連付いた署名鍵の CSP 情報を取得	×	○	×
署名鍵のコンテナ名かつ PROV_RSA_AES で CSP 再初期化	×	○	×
署名処理実行	○	○	○
※1　LpkCert の setCapi(xxx, false) または setFile()/setBin() 他の場合 ※2　LpkCert の setCapi(xxx, true) または setCard(LPK_CT_CAPI) の場合			

CAPI による署名手順の違い

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．６．CAPI（CryptoAPI）対応 IC カードによる署名の利用

IC カードの種類による証明書と署名鍵の指定。JPKI/HPKI の場合に認証パスに必要な証明書が含まれている場合は署名時に取得することも出来る。

V1.07.R2 より IC カードのスペック指定をする setCard2() が追加された。これによりスペックが分かっている IC カードの指定が可能となった。LpkCmd/LpaCmd/Lx3Cmd において 7 種類のプリセットスペックが用意された。V1.10.R1 より CRPKI リモート署名が追加され 8 種類となった。

API	利用方法
LpkCmd LpaCmd Lx3Cmd	-cert オプションの card 引数で指定 利用方法 1： -cert card <type> 例 1）LpkCmd -sign -cert card jpki … 利用方法 2： -cert card type <numb> 例 2）LpkCmd -sign -cert card type 1 –in in.pdf -out out.pdf
C++	LpkCert クラスの setCard() または setCard2() メソッドで指定 利用方法 1： int setCard(LPK_CARD_TYPE type); 利用方法 2： int setCard2(const CHAR* provName, int provType, int provFlag, int keyType, bool chain);
Java	LpkCert クラスの setCard() または setCard2() メソッドで指定 利用方法 1： int setCard(int type); 利用方法 2： int setCard2(Sring provName, int provType, int provFlag, int keyType, boolean chain);

証明書選択画面による証明書と署名鍵の指定

No	スペック（setCard2 の引数で指定）	主なカード種別
1	"JPKI Crypto Service Provider for Sign", PROV_RSA_AES, 0, -1	JPKI 署名用
2	"JPKI Crypto Service Provider for Auth", PROV_RSA_AES, 0, -1	JPKI 認証用
3	"JPKI Crypto Service Provider", PROV_RSA_FULL, 0, -1	GPKI/JPKI(旧住基カード)
4	"Melco Standard-9 Enhanced Cryptographic Service Provider", PROV_RSA_AES, 0, -1	DIACERT-P(ジャパンネット) LGPKI
5	"NEC Secure Ware AES Cryptographic Provider", PROV_RSA_AES, CRYPT_SILENT, -1	NDN(日本電子認証) [Hojin, AOSign, GoSign]
6	"DNP Standard-9 Cryptographic Service Provider", PROV_RSA_AES, 0, 1	ToiNX
7	"CRPKI Crypto Service Provider for Remote Sign ", PROV_RSA_AES, 0, -1	CRPKI リモート署名用
8	MS_SCARD_PROV_A, PROV_RSA_FULL, CRYPT_SILENT, -1	Pentio,e-Probatio,LGPKI 等

LpkCmd/LpaCmd/Lx3Cmd にプリセットされている IC カード設定と主なカード種別

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

LPK_CARD_TYPE	<type>	説明
LPK_CT_CAPI	capi	CAPI 経由の IC カード利用（GPKI の IC カード等） ※ 証明書選択画面が開かれるので自分で選択する必要あり ※ LpkCert.setCard() にて種別を設定する。 ※ LpkCert.setCapi() の第 2 引数を true にした場合と同じ
LPK_CT_JPKI	jpki	JPKI 公的個人認証サービスの IC カード利用 情報 http://www.lascom.or.jp/jinfo/spec2.html カード AP ライブラリ CryptoAPI 編（PDF ファイル） http://www.lascom.or.jp/jinfo/jpkiapv2/01CAPI.pdf ※ LpkCert.setCard() にて種別を設定する。
LPK_CT_HPKI	hpki	HPKI ヘルスケア PKI の IC カード利用 情報 http://www.jahis.jp/10-002/ HPKI 対応 IC カードガイドライン（PDF ファイル） http://www.jahis.jp/wp/wp-content/uploads/st10-002.pdf ※ LpkCert.setCard() にて種別を設定する。 ※ 実装済みだが動作未確認
LPK_CT_SPEC	spec	CSP 名やプロバイダタイプ等の詳細スペック指定 ※ LpkCert.setCard2() にて詳細スペックを設定する。 ※ V1.07.R2 よりサポート

CAPI 利用 IC カード指定の種類

内容	種類	説明
CSP 名 （プロバイダ名）	JPKI	"JPKI Crypto Service Provider for Sign" が指定される
	HPKI	"HPKI Crypto Service Provider for Non Repudiation" が指定される
認証パス証明書	JPKI	IC カードに格納された都道府県知事の自己署名証明書を取得
	HPKI	IC カードに格納された CA の証明書（チェーン）を取得 ※ SET OF Certificate; として取得

JPKI/HPKI が明示された場合の対応内容

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

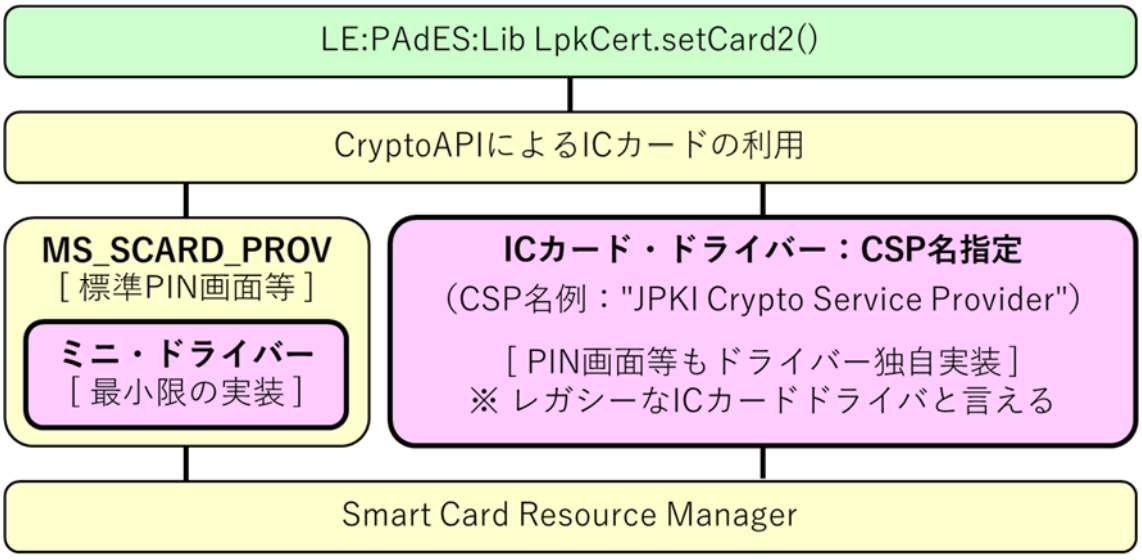
※ LPK_CT_SPEC：詳細スペック指定による IC カード利用方法（V1.07.R2 からサポート）

LPK_CT_SPEC を利用するには LpkCert.setCard2() を利用する。setCard2() は以下の 5 つの引数を持つ。ここでは各引数について解説をする。

引数	タイプ	概要
provName	文字列	プロバイダ名を指定
provType	整数値	プロバイダ種別を指定（PROV_RSA_FULL/PROV_RSA_AES 等）
provFlag	整数値	プロバイダフラグを指定（CRYPT_SILENT 等）
keyType	整数値	鍵種別を指定（AT_SIGNATURE[2]/AT_KEYEXCHANGE[1]等）
chain	ブール	IC カードの CA 証明書や証明書チェーンの取得

1）provName：プロバイダ名

Windows では CSP（Cryptographic Service Provider）名を指定することで IC カードドライバーの指定が可能となっている。CSP 名をここではプロバイダ名と呼ぶ。CSP 名は IC カード提供元から情報を取得できる。例えばマイナンバーカードの署名用プロバイダ名は "JPKI Crypto Service Provider for Sign" となっている。最近では標準化された PIN 入力画面を利用するミニ・ドライバー方式で IC カードドライバーを提供する事も増えている。ミニ・ドライバーでは CSP 名としてシステム定義されている MS_SCARD_PROV_A（"Microsoft Base Smart Card Crypto Provider"）を指定する。いずれにせよ IC カードドライバーは必要であり利用前にインストールされている必要がある。



Windows 環境における IC カードドライバーの構造
（ピンクのドライバー部が IC カードベンダーから提供される部分）

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

２）provType：プロバイダ種別

Windows では IC カードドライバーも含めて CryptoAPI 実装が複数用意されている。プロバイダの種別によっては利用可能となる暗号アルゴリズムが異なるが、IC カードドライバーでは通常は指定するプロバイダ種別も決められている。一般的には PROV_RSA_AES（24）または PROV_RSA_FULL（1）のどちらかを指定する。別途指定がある場合にはそれを指定する。

３）provFlag：プロバイダフラグ

IC カードドライバーのオプションをフラグ指定ができる。通常は指定する必要がない（0 を指定）が、CRYPT_SILENT（0x00000040）の指定をした方が良い場合がある。CRYPT_SILENT を指定することで、CryptoAPI を初期化する際に PIN 入力を求められないようにすることができる。PIN 入力は署名鍵にアクセスする時に求められる。

４）keyType：鍵種別

署名鍵の種別を指定する。通常は AT_SIGNATURE(2)を指定すれば良いが、AT_KEYEXCHANGE(1) を指定する必要がある場合がある。なお-1 を指定することでデフォルト値（AT_SIGNATURE）を利用する。

５）chain：CA 証明書や証明書チェーンの取得

オプションとして IC カードに格納されている CA 証明書や証明書チェーンを取得することができる。chain 引数を false として場合には署名証明書のみの取得となる。chain 引数を true にした場合には PP_CERTCHAIN を指定して IC カードに含まれている CA 証明書他を取得する。なおマイナンバーカードの場合（プロバイダ名が"JPKI Crypto Service Provider for"で始まる）は PP_JPKI_CA_CERTIFICATE を指定して CA 証明書を取得する。

マイナンバーカードのように特殊な指定が必要な場合もあり、うまく取得出来ない場合には chain 引数は false にして利用する。false の場合には CA 証明書は署名データに埋め込まれないが、別途長期署名化することで後から埋め込むことができる。

※ 引数例：JPKI（マイナンバーカード）署名用の場合

引数	指定	説明
provName	"JPKI Crypto Service Provider for Sign"	署名用の指定
provType	PROV_RSA_AES	必須、PROV_RSA_FULL は不可
provFlag	0	プロバイダフラグ無し
keyType	-1	デフォルト値 AT_SIGNATURE[2] の指定
chain	true	CA 証明書を取得して利用する

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．７．PKCS#12 ファイルによる署名の利用

PKCS#12 ファイル形式による証明書と署名鍵の指定は Linux 版でも利用可能となっている。ファイルパス以外にパスワード（PIN）が必須となる。PKCS#12 ファイルに認証パスに必要な証明書が含まれている場合は署名時に取得することが出来る。

API	利用方法
LpkCmd LpaCmd Lx3Cmd	-cert オプションの p12 引数で指定 利用方法： -cert p12 <path> <passwd> 例）LpkCmd -sign -cert p12 pkcs12file.p12 passwd …
C++	LpkCert クラスの setPkcs12() メソッドで指定、ファイルパスかバイナリ指定 利用方法： int setPkcs12(const CHAR* path, const CHAR* passwd); int setPkcs12(const BINARY& bin, const CHAR* passwd);
Java	LpkCert クラスの setPkcs12() メソッドで指定、ファイルパスかバイナリ指定 利用方法： int setPkcs12(String path, String passwd); int setPkcs12(byte[] bin, String passwd);

PKCS#12 形式による証明書と署名鍵の指定

※ PKCS#12 ファイルを署名用にサーバー上やデータベース上に置く運用は簡易ではあるが安全性が低いので注意が必要。可能であれば HSM の上に署名鍵を置くべきである。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．８．PKCS#11 対応 IC カードによる署名の利用

Ver1.04.R2 版より新機能として PKCS#11 のインターフェイスによる IC カードの利用に対応した。IC カードには「３．６．CAPI 対応 IC カードによる署名の利用」にて説明をした CAPI 対応の形式もあるが、PKCS#11 ドライバを使う方法も広く使われている。IC カードの利用時には API が CAPI 対応か PKCS#11 対応かを確認する必要がある。

PKCS#11 対応の IC カードを利用する為には、その IC カードに対応した PKCS#11 ドライバファイルが別途必要となる。PKCS#11 ドライバファイルの Windows 用は DLL ファイルで提供される。このドライバファイルを証明書指定時の引数として指定する必要がある。PKCS#11 は Linux 環境で利用されることもあるが、現在試験ができる環境が無い為に未対応である。なお Linux 用のドライバファイルは .so シェアードファイルで提供される。Linux 対応をご希望の場合には別途有償にて対応を検討するので問合せください。

API	利用方法
LpkCmd LpaCmd Lx3Cmd	-cert オプションの p11 引数で指定 ※ dll_file はフルパスが望ましい 利用方法： -cert p11 <dll_file> <passwd> 例）LpkCmd -sign -cert p11 pkcs11.dll passwd …
C++	LpkCert クラスの initPkcs11() メソッドで指定、ドライバの DLL ファイルパスを指定 利用方法： int initPkcs11(const CHAR* dll_file, const CHAR* passwd); ※ dll_file はフルパスが望ましい
Java	LpkCert クラスの initPkcs11() メソッドで指定、ドライバの DLL ファイルパスを指定 利用方法： int setPkcs12(String dll_file, String passwd); ※ dll_file はフルパスが望ましい

PKCS#11 の IC カードによる証明書と署名鍵の指定

PKCS#11 利用時のエラーとして新規に以下 2 つのエラーコードが追加された。

```
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
```

最後に生じた PKCS#11 のエラーメッセージは static な API である LpkCert:: getP11Mesg() により取得できる。上記 2 エラーとなる場合には LpkCert:: getP11Mesg() によりエラー情報を取得すると PKCS#11 の API 名と HEX 値のエラーコードが確認できる場合があるが、このエラーコードは local¥include¥pkcs11¥pkcs11t.h に記載されている。例えば「C_OpenSession failed (0x00000030)」と言うエラーメッセージの場合には C_OpenSession にて CKR_DEVICE_ERROR (0x00000030)のエラーを生じたと言う意味であり、IC カードが挿入されていない等の意味となる。詳しくは CKR_DEVICE_ERROR で検索してみると良い。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．９．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、証明書検証サーバー（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

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

なお取得した 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

※ LE:PAdES:Lib や LE:XAdES:Lib にて CVS を使う場合にはそれぞれのマニュアルに使い方を記載している。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．１０．LE:PKI:Lib 試験用環境（証明書やタイムスタンプ等）

動作サンプル（sample/LePKI/ や sample/LePAdES/ や sample/LeXAdES3/フォルダの下）ではラング・エッジの試験用 PKI 環境を利用している。ラング・エッジ製品を使った開発時には自由に使える。なお最新の証明書や情報はラング・エッジの試験用認証局リポジトリで公開している。

ラング・エッジ試験用認証局リポジトリ
https://www.langedge.jp/democa/

１）試験用ルート証明書（トラストアンカー）

現在ルート証明書は CA1 と CA2 の 2 種類を提供している。CA1 は署名証明書用として、CA2 は TSA（タイムスタンプ）証明書として利用している。store/trusts フォルダの下に格納されている。なお Acrobat 等の検証の為にこれらの**トラストアンカーを Windows 証明書ストアの「信頼されたルート証明機関」に入れることはセキュリティ上のリスクを伴うので自己責任においてインストールすること**。またインストールする場合にも以下の指紋を確認すること。

Common Name	ファイル名	指紋（SHA-1 ハッシュ値）
LangEdge CA Root 001	le-test-root001.cer	2ae08369f3c0a932fb4711f9de11823af97b0c5a
LangEdge CA Root 002	le-test-root002.cer	088f6a36a9875ea68bdc3a2efd663e2725a90df6

第 3 世代 試験用ルート証明書一覧（2023/01/13～2043/01/28）

Common Name	ファイル名	指紋（SHA-1 ハッシュ値）
LangEdge CA Root 01	le-test-root01.cer	5d7dabed378a60a3047d42be53587667401d9332
LangEdge CA Root 02	le-test-root02.cer	51f0acf8b3e94083c5d33e59d48f9590fcfb0679

第 2 世代 試験用ルート証明書一覧（2018/04/14～2038/04/29）

Common Name	ファイル名	指紋（SHA-1 ハッシュ値）
LangEdge CA Root 1	le-test-root1.cer	10fe2d184b3a11b84d41458df61435595820fde7
LangEdge CA Root 2	le-test-root2.cer	bee5fc5cbdf954923b661945dad32aefe620c9ab

第 1 世代 試験用ルート証明書一覧（2013/04/16～2018/04/20）

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

2）試験用証明書（署名鍵付き）と CRL 発行

試験用証明書のうち署名に利用する PKCS#12 ファイル（拡張子 .p12）を sample/LePKI フォルダの下に 2 種類提供している。「LeRevoke.p12」は失効済み証明書であり失効検証試験に利用ができる。通常は「LeTest.p12」を使って動作試験を行う。パスワードは ”test” である。TSA 証明書はラング・エッジのサーバー内の署名で利用される為に署名鍵は LE:PKI:Lib では提供されない。

Common Name	CA	ファイル名	指紋（SHA-1 ハッシュ値）	用途
LE Test2 100017	1	LeTest.p12	b892ab99cde4a8eb925176fd6cf29320ece4fb53	試験証明書
LE Test2 100018	1	LeRevoke.p12	9de3e151fc778bf14d1362bc0392106de91bf965	失効証明書
LE TSA 200020	2	---	8f3a09936b6fca1d407f08e6121ec66b20cd89d7	TSA 証明書

第 3 世代 試験用証明書一覧（2023/01/14～2030/01/01）

いずれも CA1/CA2 のルート証明書により署名された CRL が発行される。毎晩更新されるので猶予期間の動作確認にも利用できる。各証明書の CRLDP にセットされている。

CA	CRLDP（CRL 発行ポイント）	更新頻度	注意
1	https://www.langedge.jp/democa/ca5.crl	毎日午前 4 時更新	24 時間稼働は保証されないので障害で更新されない場合もある
2	https://www.langedge.jp/democa/ca6.crl		

第 3 世代 試験証明書用 CRL 発行一覧

CA	CRLDP（CRL 発行ポイント）	更新頻度	注意
1	http://www.langedge.jp/democa/ca3.crl	毎日午前 4 時更新	24 時間稼働は保証されないので障害で更新されない場合もある
2	http://www.langedge.jp/democa/ca4.crl		

第 2 世代 試験証明書用 CRL 発行一覧

CA	CRLDP（CRL 発行ポイント）	更新頻度	注意
1	http://www.langedge.jp/democa/ca1.crl	毎日午前 4 時更新	24 時間稼働は保証されないので障害で更新されない場合もある
2	http://www.langedge.jp/democa/ca2.crl		

第 1 世代 試験証明書用 CRL 発行一覧

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

3）試験用タイムスタンプサービス

試験用タイムスタンプサービスの URL は認証無しの demotsa と Basic 認証の batsa の 2 種類が公開されている。URL は異なるがタイムスタンプサービスとしては 1 つとなっている。OpenSSL 1.0.0 以降の openssl コマンドの ts オプションによりタイムスタンプ応答を生成して返している。応答時刻は ntp により NICT に接続をして同期しているので秒単位のずれはあるので注意が必要である。もちろん本番稼働時には利用しないこと。あくまで**開発時の試験用であり正式には必要に応じて商用タイムスタンプサービス等を利用すること**。ハッシュ方式としては SHA-1/SHA-256/SHA-512 に対応している。現在ポリシとして 2.5.29.32.0（anyPolicy）がセットされている。時刻監査証明書（属性証明書）は入っていない。

CA	TSA の URL	認証	認証情報
2	https://www.langedge.jp/tsa https://www.langedge.jp/demotsa	無し	---
2	https://www.langedge.jp/batsa	Basic 認証	UserID "langedge" / Password "test"

試験用 TSA 一覧

なお公開しているタイムスタンプサーバーはラング・エッジで独自開発した FreeTSA を利用している。以下に FreeTSA の情報があるので、自分で試験用のタイムスタンプサーバーを構築することも可能となっている。

フリータイムスタンプ局（FreeTSA）のすゝめ
https://www.langedge.jp/blog/index.php?itemid=665

10 分でできるタイムスタンプ局（FreeTSA Project）
https://www.langedge.jp/blog/index.php?itemid=664

自由に使えるタイムスタンプ局（FreeTSA Service）
https://www.langedge.jp/blog/index.php?itemid=663

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．１１．失効情報取得の高度な指定

失効情報 CRL と OCSP の扱いは組み合わせも含めて運用上の注意が必要である。現在検討可能な機能として以下の 3 機能がある。

1. CRL/OCSP の優先指定（**※ V1.08.R1 にて仕様変更注意**）
2. 取得失効情報の独自証明書ストアへの追加機能
3. 独自検証プロキシサーバーの利用機能

１）CRL/OCSP の優先指定

証明書の種類によっては CRL 配布点と OCSP アドレスの両方が記載されている場合がある。この場合に長期署名の場合は CRL と OCSP のどちらを利用するかを判断する必要がある。なお長期署名を使わない通常の署名であればどちらを選択しても問題は無い。

署名証明書	アーカイブタイムスタンプがすぐに打たれるのであれば CRL と OCSP のどちらを選んでも良い。OCSP であれば一般には失効がすぐに反映されるので OCSP 優先とすることで猶予期間（CRL 発行待ち）が不要となる利点がある。 **推奨：OCSP 優先（V1.08.R1 のデフォルト設定）**
TSA 証明書	TSA 証明書よりも OCSP レスポンダ証明書の方の有効期間が短い場合に、10 年では無く OCSP レスポンダの有効期間が優先されてしまう。CRL の標準仕様では TSA 証明書と同じ親証明書となるのでこの問題は生じない。ただし OCSP であれば一般には失効がすぐに反映されるので OCSP 優先とすることで猶予期間（CRL 発行待ち）が不要となる利点がある。 **推奨：OCSP 優先（V1.08.R1 のデフォルト設定）** **ただし OCSP レスポンダ証明書の有効期間が短い場合には CRL 優先が望ましい。**

なお OCSP レスポンダ証明書の有効期限が短い場合であっても、検証後すぐにアーカイブタイムスタンプを打つ仕様であれば問題は無い。

V1.08.R1 からはデフォルトが OCSP 優先へと変更になった。もし V1.07 以前と同じく CRL 優先にしたい場合には「CRL を優先」設定のフラグ（LPK_VERIFY_FLAG::LPKV_PRIOR_CRL）を使うことで CRL 優先となる。なお LpkCmd/LpaCmd/Lx3Cmd では -prior 引数で指定する。

API	検証時の検証フラグに LPKV_PRIOR_CRL を指定することで CRL 優先に出来る。
LpkCmd/ LpaCmd/Lx3Cmd	-prior crl ： 失効情報優先フラグ指定 crl ： CRL 優先、省略時は OCSP 利用

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

２）取得失効情報の独自証明書ストアへの追加機能

従来は CRL キャッシュ機能（ファイル保存）を提供していたが、スレッド実行時に同時アクセスする場合に問題があるケースがあった。この為に取得した失効情報（CRL/OCSP）をメモリ上で保持してキャッシュする機能を V1.06.R1 より提供する。情報を保持するのは LePKI インスタンスであるので、同じ LePKI インスタンス（または LePAdES/LeXAdES3 インスタンス）を保持したまま複数ファイルの検証を行うことで、失効情報をメモリ上にキャッシュして再利用する。

指定は検証フラグ（LPK_VERIFY_FLAG::LPKV_ADD_VALID）を指定するか LpaCmd/Lx3Cmd の -addvalid を利用する。

API	検証時の検証フラグに LPKV_ADD_VALID を指定する。
LpaCmd/Lx3Cmd	-addvalid ： 取得した検証情報(CRL/OCSP)を独自ストアに追加する

※　CRL キャッシュ機能は V1.08.R1 より未サポートとなった。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

3）独自検証プロキシサーバーの利用機能

検証時に CRL や OCSP は指定されたサーバーに HTTP 通信にて取得に行く。これを決められた独自の検証プロキシサーバー経由で取得する機能が V1.06.R1 より追加された。これによりファイアウォール等で接続先に制限がある環境でも失効情報を取得することができるようになった。

独自検証プロキシサーバーの設定は LePKI::addRevoProxyServer() で指定する。また LpaCmd では-revoproxy 引数で指定する。

API	LePKI::addRevoProxyServer(URL); で指定する。
LpkCmd/ LpaCmd/Lx3Cmd	-revoproxy url　: LE 独自検証プロキシサーバーの指定

独自検証プロキシサーバーが指定された場合には、CRL/OCSP 取得は独自のヘッダキー名で転送先を指定して呼び出される。

キー名	意味	例	補足
X-LE-CRL-Location	CRL 取得 URL	http://dev.langedge.jp/test.crl	GET 時
X-LE-OCSP-Location	OCSP 取得 URL	http://dev.langedge.jp/ocsp	POST 時

独自のヘッダキー設定

独自の検証プロキシサーバーは別途用意する必要がある。参考用として Java の Tomcat 用の実装サンプルを sample/LePKI-server/LeRevoProxy の下に提供している。

ファイル名	概要
LePkiRevo.java	LE 独自検証プロキシサーバーのサンプルソース
HttpUtil.java	LePkiRevo から使われる HTTP 通信部のサンプルソース
build.sh	LePkiRevo をビルドする Linux 用のサンプルシェルスクリプト

なお検証フラグ（LPK_VERIFY_FLAG::LPKV_PROXY_RETRY）を使うことで、独自検証プロキシサーバーでの情報取得に失敗した場合に、元のアドレスを使って再取得することも可能である。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．１２．CRL キャッシュ機能

※　CRL キャッシュ機能は V1.08.R1 より未サポートとなった。

３．１３．ECDSA（楕円曲線暗号）対応

V1.09.B1 より署名方式として ECDSA（楕円曲線暗号）署名方式に対応した。V1.09.B1 では PKCS#11 の ECDSA 署名には未対応である。PKCS#12 または Windows 証明書ストア経由にて ECDSA 署名が可能となっている。また PKCS7 形式の署名データでは ECDSA 署名には未対応（署名時に LPK_ERR_NO_SUPPORTED が返る）となる。PKCS7 では検証のみ ECDSA 署名に対応している。

ECDSA の署名値としては、DER 形式・BER 形式・バイナリ形式の 3 種類があるが、LePKI の内部では全て DER 形式に変換して利用している。なお ECDSA の署名値は 2 つの BigInteger（整数）で構成されている。

ECDSA 署名値	説明
DER 形式	最も一般的であり OpenSSL 等の標準形式。
BER 形式	BigInteger が DER 正規化されていない形式。OpenSSL 等では検証不可。
バイナリ形式	2 つの同じサイズの BigInteger を単純に結合した形式。 Windows の CNG による ECDSA 署名した場合の形式。

ECDSA 署名値の形式

証明書クラス LpkCert に公開鍵の署名方式を取得する getKeyAlg() を追加した。公開鍵が RSA 方式なら LPK_RSA が、ECDSA 方式なら LPK_ECDSA が返される。LpkeCades と LpkPkcs7 クラスには getSignType() が追加され、署名暗号方式 LPK_SIGN が返される。

なお検証において CRL に関しては ECDSA 署名に対応済みとなる。OCSP に関して、実装は完了しているが試験する環境が未整備である為に V1.09.B1 では未試験となっている。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

３．１４．CNG（Cryptography API: Next Generation）対応 IC カードによる署名の利用

V1.10.R1 より IC カードの CNG スペック指定をする setCardCng() が追加された。これによりスペックが分かっている CNG 対応の IC カードの指定が可能となった。LpkCmd/LpaCmd/Lx3Cmd において、仮想 IC カードとして提供される商業登記電子証明書（CRPKI）のリモート署名ドライバソフトのプリセットスペックが用意された。なお CRPKI の CNG 対応は 64 ビット版のみ提供されており、32 ビット版の LE:PKI:Lib では署名ができないので注意が必要となる。

API	利用方法
LpkCmd LpaCmd Lx3Cmd	-cert オプションの card 引数で指定 利用方法： -cert card cng <numb> 例）LpkCmd -sign -cert card cng 1 –in in.pdf -out out.pdf
C++	LpkCert クラスの setCardCng() メソッドで指定 利用方法： int setCardCng(const CHAR* provName, int provType, int provFlag, int keyType, bool chain);
Java	LpkCert クラスの setCardCng() メソッドで指定 利用方法： int setCardCng(Sring provName, int provType, int provFlag, int keyType, boolean chain);

証明書選択画面による証明書と署名鍵の指定

No	スペック（setCardCng の引数で指定）	主なカード種別
1	"CRPKI Key Storage Provider for Remote Sign", NULL, 0	CRPKI リモート署名用

LpkCmd/LpaCmd/Lx3Cmd にプリセットされている CNG の IC カード設定と主なカード種別

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

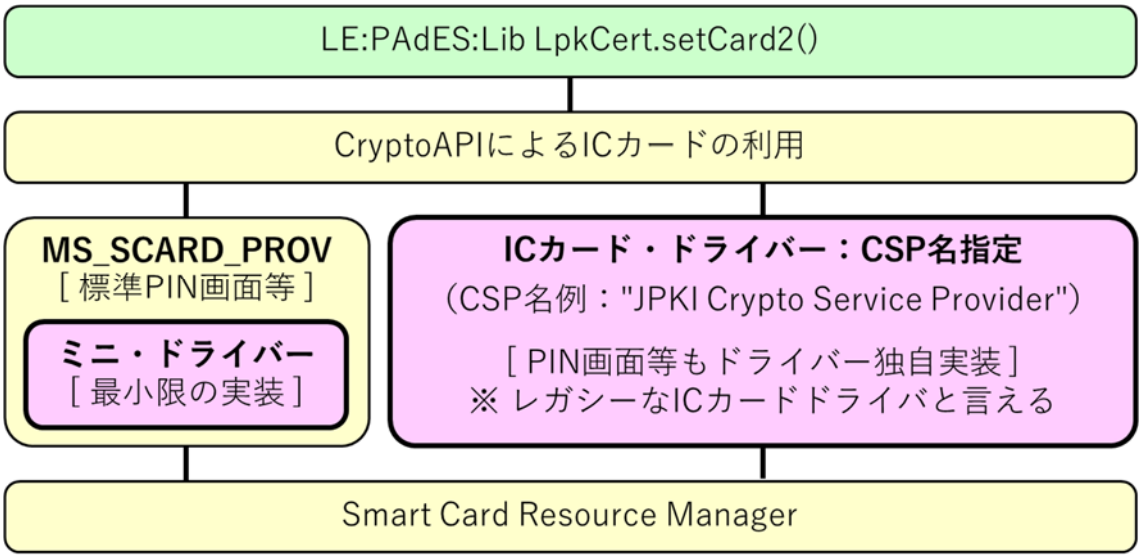
※ LPK_CT_SPEC：詳細スペック指定による IC カード利用方法（V1.07.R2 からサポート）

LPK_CT_SPEC を利用するには LpkCert.setCard2() を利用する。setCard2() は以下の 5 つの引数を持つ。ここでは各引数について解説をする。

引数	タイプ	概要
provName	文字列	プロバイダ名を指定
provType	整数値	プロバイダ種別を指定（PROV_RSA_FULL/PROV_RSA_AES 等）
provFlag	整数値	プロバイダフラグを指定（CRYPT_SILENT 等）
keyType	整数値	鍵種別を指定（AT_SIGNATURE[2]/AT_KEYEXCHANGE[1]等）
chain	ブール	IC カードの CA 証明書や証明書チェーンの取得

1）provName：プロバイダ名

Windows では CSP（Cryptographic Service Provider）名を指定することで IC カードドライバーの指定が可能となっている。CSP 名をここではプロバイダ名と呼ぶ。CSP 名は IC カード提供元から情報を取得できる。例えばマイナンバーカードの署名用プロバイダ名は "JPKI Crypto Service Provider for Sign" となっている。最近では標準化された PIN 入力画面を利用するミニ・ドライバー方式で IC カードドライバーを提供する事も増えている。ミニ・ドライバーでは CSP 名としてシステム定義されている MS_SCARD_PROV_A（"Microsoft Base Smart Card Crypto Provider"）を指定する。いずれにせよ IC カードドライバーは必要であり利用前にインストールされている必要がある。



Windows 環境における IC カードドライバーの構造
（ピンクのドライバー部が IC カードベンダーから提供される部分）

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

２）provType：プロバイダ種別

Windows では IC カードドライバーも含めて CryptoAPI 実装が複数用意されている。プロバイダの種別によっては利用可能となる暗号アルゴリズムが異なるが、IC カードドライバーでは通常は指定するプロバイダ種別も決められている。一般的には PROV_RSA_AES（24）または PROV_RSA_FULL（1）のどちらかを指定する。別途指定がある場合にはそれを指定する。

３）provFlag：プロバイダフラグ

IC カードドライバーのオプションをフラグ指定ができる。通常は指定する必要がない（0 を指定）が、CRYPT_SILENT（0x00000040）の指定をした方が良い場合がある。CRYPT_SILENT を指定することで、CryptoAPI を初期化する際に PIN 入力を求められないようにすることができる。PIN 入力は署名鍵にアクセスする時に求められる。

４）keyType：鍵種別

署名鍵の種別を指定する。通常は AT_SIGNATURE(2)を指定すれば良いが、AT_KEYEXCHANGE（1）を指定する必要がある場合がある。なお-1 を指定することでデフォルト値（AT_SIGNATURE）を利用する。

５）chain：CA 証明書や証明書チェーンの取得

オプションとして IC カードに格納されている CA 証明書や証明書チェーンを取得することができる。chain 引数を false として場合には署名証明書のみの取得となる。chain 引数を true にした場合には PP_CERTCHAIN を指定して IC カードに含まれている CA 証明書他を取得する。なおマイナンバーカードの場合（プロバイダ名が"JPKI Crypto Service Provider for"で始まる）は PP_JPKI_CA_CERTIFICATE を指定して CA 証明書を取得する。

マイナンバーカードのように特殊な指定が必要な場合もあり、うまく取得出来ない場合には chain 引数は false にして利用する。false の場合には CA 証明書は署名データに埋め込まれないが、別途長期署名化することで後から埋め込むことができる。

※ 引数例：JPKI（マイナンバーカード）署名用の場合

引数	指定	説明
provName	"JPKI Crypto Service Provider for Sign"	署名用の指定
provType	PROV_RSA_AES	必須、PROV_RSA_FULL は不可
provFlag	0	プロバイダフラグ無し
keyType	-1	デフォルト値 AT_SIGNATURE[2] の指定
chain	true	CA 証明書を取得して利用する

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

４．ネットワーク機能

本章ではネットワークの機能、特に HTTP/HTTPS 通信に関する説明をする。ネットワーク機能は特に PKI において、タイムスタンプ取得・OCSP 取得・CRL 取得・証明書取得等で使われている。CRL や証明書の取得では LDAP 通信（LDAPS 通信はほぼ使われていない）も使われる事がある。

HTTP 通信	LE:PKI:Lib での対応	補足
Socket 利用 (独自実装)	Linux 版にて利用 Ver1.08.R1 から Windows 版でも利用可能 ※ Linux 版では Socket のみ利用可能	HTTPS（TLS）通信も OpenSSL を使っているので利用可能
WinHTTP 利用	Windows Update でも使われている標準の HTTP 通信モジュール ※ Windows 版 Ver1.08.R1 デフォルト利用	Linux 版では利用不可
WinInet 利用	IE の HTTP 通信モジュール	Linux 版では利用不可 今後の利用は非推奨

HTTP 通信に利用している実装

LDAP 通信	LE:PKI:Lib での対応	補足
OpenLDAP 利用	Linux 版にて利用 Windows 版デフォルト利用	現在 LDAPS（TLS）通信には非対応 Windows 版では CAPI にて LDAPS 利用可能
WinLdap 利用 (CAPI も利用)	Windows 版で補助利用	OpenLDAP 通信に失敗した場合に利用される Linux 版では利用不可

LDAP 通信に利用している実装

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

４．１．Windows 版：通信方式の指定

Windows 版では Ver1.08.R1 から新たに LPK_HTTP_TYPE が定義された。通常はデフォルト設定の LPK_HTTP_DEFALT がセットされている。デフォルト設定は Windows 版であれば LPK_HTTP_WINHTTP が、Linux 版であれば LPK_HTTP_SOCKET がセットされている。明示的に指定したい場合には LePKI や LpkUtil の該当 API の引数で指定が可能となっている。Ver1.08.R1 以降であっても WinInet を利用したい場合には LPK_HTTP_WININET を指定することで、従来通り WinInet の通信も可能となっている。

```
/** HTTP 通信種別（タイムスタンプ/CRL/OCSP 等の取得）*/
typedef enum {
    LPK_HTTP_DEFALT  = -1,  ///< LPK_HTTP_DEFALT: デフォルト設定（Win:WinHTTP/Linux:Socket）
    LPK_HTTP_WININET = 0,   ///< LPK_HTTP_WININET: WinInet（Windows 環境のみ指定可）
    LPK_HTTP_WINHTTP = 2,   ///< LPK_HTTP_WINHTTP: WinHTTP（Windows 環境のみ指定可）
    LPK_HTTP_SOCKET  = 1    ///< LPK_HTTP_SOCKET: Socket（OpenSSL 利用）
} LPK_HTTP_TYPE;
```

HTTP 通信種別の定義

```
    LePKI.setHttp(LPK_HTTP_TYPE htype);          // 検証時の CRL/OCSP 取得の HTTP 方式の指定
    LpkTimestamp.setHttp(LPK_HTTP_TYPE htype);   // タイムスタンプ取得時の HTTP 方式の指定
    LpkUtil.getCrl(..., LPK_HTTP_TYPE htype);    // CRL 取得時の HTTP 方式の指定
    LpkUtil.getOcsp(..., LPK_HTTP_TYPE htype);   // OCSP 取得時の HTTP 方式の指定
    LpkUtil.getOcsp2(..., LPK_HTTP_TYPE htype);  // 独自 OCSP 取得時の HTTP 方式の指定
    LpkUtil.getCvs(..., LPK_HTTP_TYPE htype);    // CVS 検証結果取得時の HTTP 方式の指定
```

Windows 版において HTTP 通信方式を指定可能な API

```
    -http <http|inet|sock> : HTTP 通信 API 種別の指定（省略時は http）
       http : WinHTTP 利用（推奨/省略時設定）
       inet : WinInet 利用（非推奨/過去互換）
       sock : 独自 Socket 利用（OpenSSL 利用）
```

Windows 版のコマンド（LpkCmd/LpaCmd/Lx3Cmd）での HTTP 通信方式の指定

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

４．２．HTTP 通信のプロキシ設定

企業内サーバーでは時にプロキシを経由したネットワーク通信が必須となる場合がある。現在 LE:PKI:Lib では認証無しのプロキシを利用した HTTP/HTTPS 通信に対応している。Socket 通信（Linux 環境）では Ver1.04.R1 からの対応となる。

１）Socket 利用時のプロキシ設定（LE_HTTP_SOCKET）

Ver1.05.R1 からプロキシ設定の API を LePKI クラスに用意した。機能自体は後述する proxy.ini ファイル利用と同じ。Ver1.08.R1 より Windows 版においても Socket 利用が可能となった。

```
/** 共通プロキシ設定の初期化(オプション)¥n¥n
 *
 * @param proxyHost [IN] 利用するプロキシサーバーの URL を指定(NULL 指定でプロキシ設定をクリア)
 * @param proxyType [IN] 利用するプロキシサーバーの種類を指定(SSL 利用時の挙動が異なる)
 * @note Socket を利用した Linux 環境用であり、Windows 環境では IE 設定を利用する
 * @note Linux 環境では認証ありの Proxy には現在未対応
 */
static void initProxy(const char* proxyHost, LPK_PROXY_TYPE proxyType);
```

```
/** Proxy(プロキシサーバー)種別.
 */
typedef enum {
        LPK_PROXY_SQUID          = 0,     ///< Squid によるプロキシサーバー(SSL が間接指定)
        LPK_PROXY_APACHE = 1,             ///< Apache によるプロキシサーバー(SSL が間接指定)
} LPK_PROXY_TYPE;
```

LePKI クラスにおけるプロキシ API

Ver1.04.R1 よりプロキシ経由の HTTP/HTTPS 通信に対応した。設定には proxy.ini ファイルを利用する。proxy.ini ファイルの置き場所は以下のいずれかとなる。

１：実行モジュール(LpaCmd 等)と同じディレクトリ下の proxy.ini
２：/usr/local/LePAdES/proxy.ini（/usr/local/LePAdES 下固定 ※1）
３：Windows 版は LE_PROXY_INIPATH 環境変数により指定された proxy.ini 利用 ※2

※1 LeXAdE3 でも LePAdES を利用
※2 本配置方法を使用できるのは、Ver1.09.R2 以降の Windows 版となる。

○ Windows 版の環境変数による設定項目（LE_HTTP_SOCKET）

環境変数	説明
LE_PROXY_INIPATH	proxy.ini ファイルのパスを指定（Windows 版 Ver1.09.R2 以降利用可） 設定例："c:¥test_setup¥proxy.ini"

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

proxy.ini ファイル内の設定はプロキシサーバーの種類により異なる。良く使われるプロキシとしては以下の 2 種類がある。この 2 種類では HTTPS 通信時の挙動が異なる。

Ａ：Apache に mod_proxy を組み込んで運用されるプロキシサーバー
Ｂ：Squid により運用されるプロキシサーバー

proxy.ini の設定は現在 1 行だけとなる。Squid プロキシを利用している場合には単にプロキシの URL をセットする。Apache の mod_proxy を使ったプロキシを利用している場合には URL の前に”A:”を追加する。以下に例を示す。

Ａ：Apache 等のプロキシサーバー利用時＝最初に"A:"を付けてプロキシサーバーの URL を指定
例 A:http://218.59.144.95:81/　　※ この例の IP アドレスは利用できない可能性あり

Ｂ：Squid 等のプロキシサーバー利用時＝プロキシサーバーの URL を指定
例 http://133.242.232.124:3128/　　※ この例の IP アドレスは利用できません

なお Apache の mod_proxy を利用する場合に、Squid と同じ設定（Ｂ：）を行った場合には、HTTP 通信は可能だが、HTTPS 通信時にエラーとなる。

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

2）WinHTTP 利用時のプロキシ設定（LE_HTTP_WINHTTP：Windows 版のみ）

WinHTTP は Windows Update にも使われている。この為にプロキシ設定は Windows システムの設定で良い。設定の「ネットワークとインターネット」＞「プロキシ」の「プロキシサーバーを使う」の「詳細」により手動による設定も可能となっている。

現在認証付きのプロキシ接続に対応する場合には、以下の環境変数としてユーザ名とパスワードをセットする。なお現在は Basic 認証のみに対応している。NTLM/PASSPORT/DIGEST 等の認証が必要な場合は別途対応が必要となるのでご相談ください。Ver1.09.R2 より設定に proxy.ini ファイルを利用したプロキシ設定に対応した。proxy.ini ファイルの置き場所は以下のいずれかとなる。

1 ：実行モジュール(LpaCmd 等)と同じディレクトリ下の proxy.ini
2 ：Windows 版は LE_PROXY_INIPATH 環境変数により指定された proxy.ini 利用 ※

※ 本配置方法を使用できるのは、Ver1.09.R2 以降の Windows 版となる。

proxy.ini の設定は現在 1 行だけで、プロキシの URL を指定する。
例 http://133.242.232.124:3128/ ※ この例の IP アドレスは利用できません

○ Windows 版の環境変数による設定項目（LE_HTTP_WINHTTP）

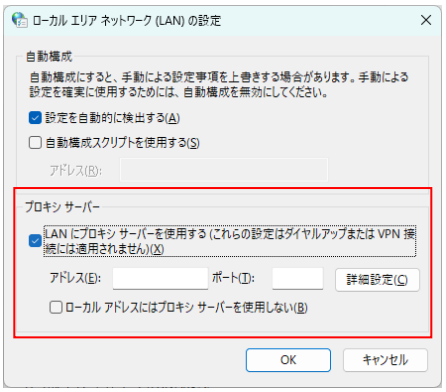
環境変数	説明
LE_PROXY_USERNAME	プロキシのユーザ名
LE_PROXY_PASSWORD	プロキシのパスワード
LE_PROXY_INIPATH	proxy.ini ファイルのパスを指定（Windows 版 Ver1.09.R2 以降利用可） 設定例："c:¥test_setup¥proxy.ini"

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

3）WinInet 利用時のプロキシ設定（LE_HTTP_WININET：Windows 版のみ）

WinInet は本来 IE（InternetExplorer）のモジュールであるので、IE のプロキシ設定がそのまま利用される。プロキシ設定は基本的にはログインしているユーザ毎に設定が必要となるので、特に ASP 環境等では注意が必要となる。PC 毎に設定をする場合は以下の手順で可能となる。

※ インターネットのプロパティから設定する

1	管理者権限のあるユーザでログオンする。
2	[ファイル名を指定して実行] で [inetcpl.cpl] 入力して[OK]することで[インターネットのプロパティ] を起動する。 ※ Windows キー＋R の同時押しで［ファイル名を指定して実行］の表示が可能。
3	[接続] タブを開き、[LAN の設定] ボタンをクリックして、[プロキシ サーバーを使用する] にチェックを入れ［アドレス］と［ポート］を入力する。 
4	[OK] と [OK] で画面を閉じる。

※ レジストリを直接編集（スクリプトやポリシーで使う場合）

レジストリキー：

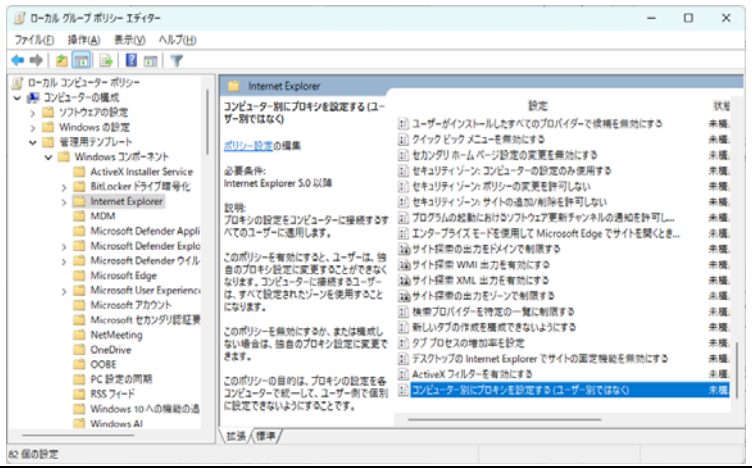
HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Internet Settings

値：

名前	種別	設定例
ProxyEnable	DWORD	1 で有効、0 で無効
ProxyServer	文字列	proxy.example.com:8080
ProxyOverride	文字列	localhost;*.example.com

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

※ ローカルグループポリシーエディタから設定のリセット（必要に応じて）

1	管理者権限のあるユーザでログオンする。
2	[ファイル名を指定して実行] で [gpedit.msc] 入力して[OK]することで[ローカルグループポリシーエディタ] を起動する。 ※ Windows キー＋R の同時押しで [ファイル名を指定して実行] の表示が可能。
3	[コンピュータの構成] - [管理用テンプレート] - [Windows コンポーネント] - [Internet Explorer] を開き、[コンピュータ別にプロキシを設定する (ユーザ別ではなく)] を有効にする。 ※ 項目のダブルクリックで編集可能になるので有効にセットする。 
4	IE（InternetExplorer）プロキシ設定がリセットされるので再度設定する。

現在認証付きのプロキシ接続に対応する場合には、以下の環境変数としてユーザ名とパスワードをセットする。また Ver1.09.R2 より設定に proxy.ini ファイルを利用したプロキシ設定に対応した。proxy.ini ファイルの置き場所は以下のいずれかとなる。

1 ：実行モジュール(LpaCmd 等)と同じディレクトリ下の proxy.ini
2 ：Windows 版は LE_PROXY_INIPATH 環境変数により指定された proxy.ini 利用 ※

※ 本配置方法を使用できるのは、Ver1.09.R2 以降の Windows 版となる。

proxy.ini の設定は現在 1 行だけで、プロキシの URL を指定する。
例 http://133.242.232.124:3128/ ※ この例の IP アドレスは利用できません

○ Windows 版の環境変数による設定項目（LE_HTTP_WININET）

環境変数	説明
LE_PROXY_USERNAME	プロキシのユーザ名
LE_PROXY_PASSWORD	プロキシのパスワード
LE_PROXY_INIPATH	proxy.ini ファイルのパスを指定（Windows 版 Ver1.09.R2 以降利用可） 設定例："c:¥test_setup¥proxy.ini"

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

付録Ａ．エラーコード

Ａ．１．LeUtil エラーコード（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．LpkCmd エラーコード（LpkCmd.h）-1200 ～ -1399

LpkCmd のエラーコードはコマンドライン LpkCmd 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_VERIFY_INDETERM = -1250,        // 検証結果が不明
LPC_ERR_VERIFY_INVALID  = -1251,        // 検証結果が不正
LPC_ERR_NOT_SUPPORT     = -1390,        // LpkCmd 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．LePKI エラーコード（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。証明書ストア等の環境を確認してください。

```
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
```

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

```
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_WCERT_CNG        = -3205, // Windows 証明書 CNG 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
```

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

```
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
```

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.

付録Ｂ．コピーライト表示

```
[OpenSSL License]
This product includes softwares developed by:
 Copyright (c) 1998-2016 The OpenSSL Project.  All rights reserved.
 This product includes software developed by the OpenSSL Project for use in the OpenSSL
Toolkit. (http://www.openssl.org/)
 This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)
 This product includes software written by Tim Hudson (tjh@cryptsoft.com)

[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/
```

※ 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 http://www.langedge.jp/

以上

Copyright © 2012-2026 LangEdge,Inc. All rights reserved.