LEクライアント署名V2.1
Edge/Chrome（カスタムURLスキーム）対応版

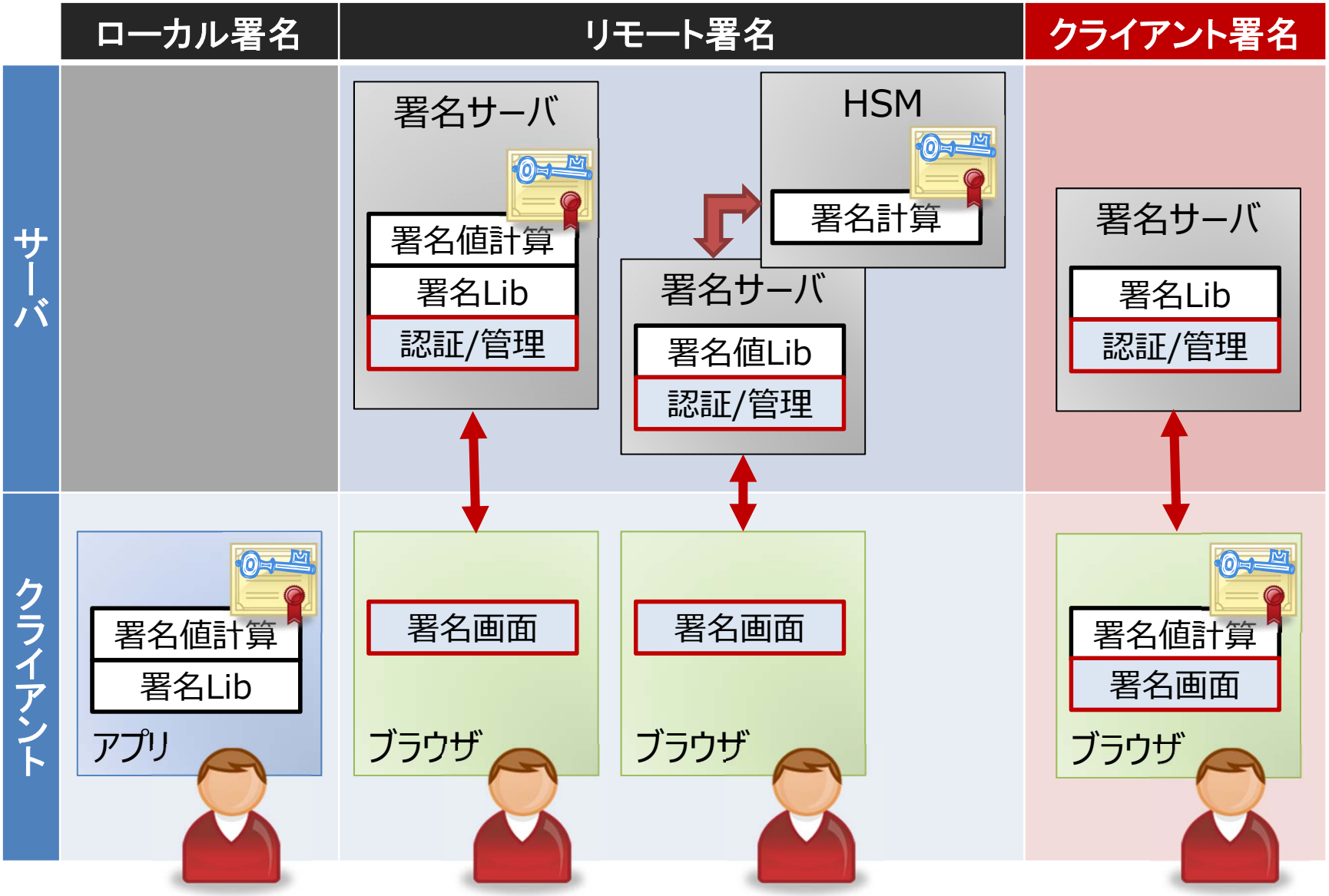
LE:Client:Sign
LangEdge クライアント 署名拡張

LangEdge,Inc.
有限会社 ラング・エッジ

プログラマ/取締役
宮地直人（miyachi@langedge.jp）

2021年1月15日

参考：一般の電子署名の実現方法



参考：一般の電子署名の実現方法の分類

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

3. クライアント署名（サーバ連携）

- ➢ サーバとクライアントが**連携**して署名を行う
- ✓ サーバ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

参考：一般的なクライアント署名の実現方法

1. ActiveXコンポーネント（IEのみ対応）

- ブラウザの中で動作するネイティブな署名プラグイン。
- 自動ダウンロードによるインストールと実行が可能。

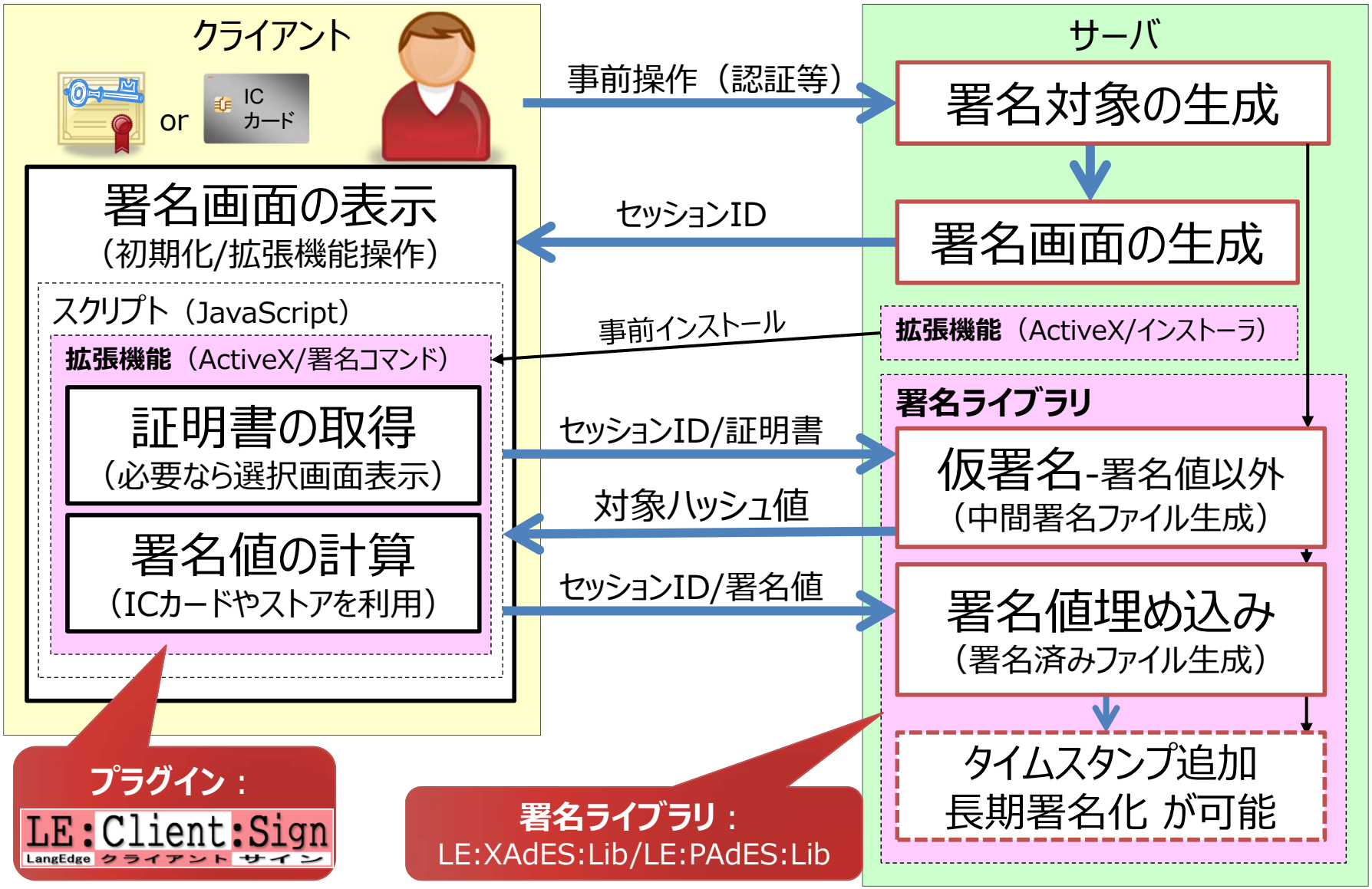
2. カスタムURLスキーム（Edge/Chrome対応）

- OSレベルで実現する任意の外部アプリ呼び出し。
- 署名アプリの事前インストールが必要。
- 署名アプリのインストーラは任意の場所で配布可能。

3. Native Messaging 拡張機能（Edge/Chrome対応）

- ブラウザの拡張機能からの外部アプリ呼び出し。
- 拡張機能の事前インストールが必要。
- 拡張機能は専用ストアへの登録と配布が必要。
- 署名アプリが別途必要になるケースもあり。

ラング・エッジ クライアント署名 イメージ図



ラング・エッジ クライアント署名 Ver2 機能

	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WebScoket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	

- NativeMessaging拡張機能には現在未対応です。
- MacOS/スマホには現在未対応です。

※ 別途開発をすることで対応する可能性はありますのでお問合せください。

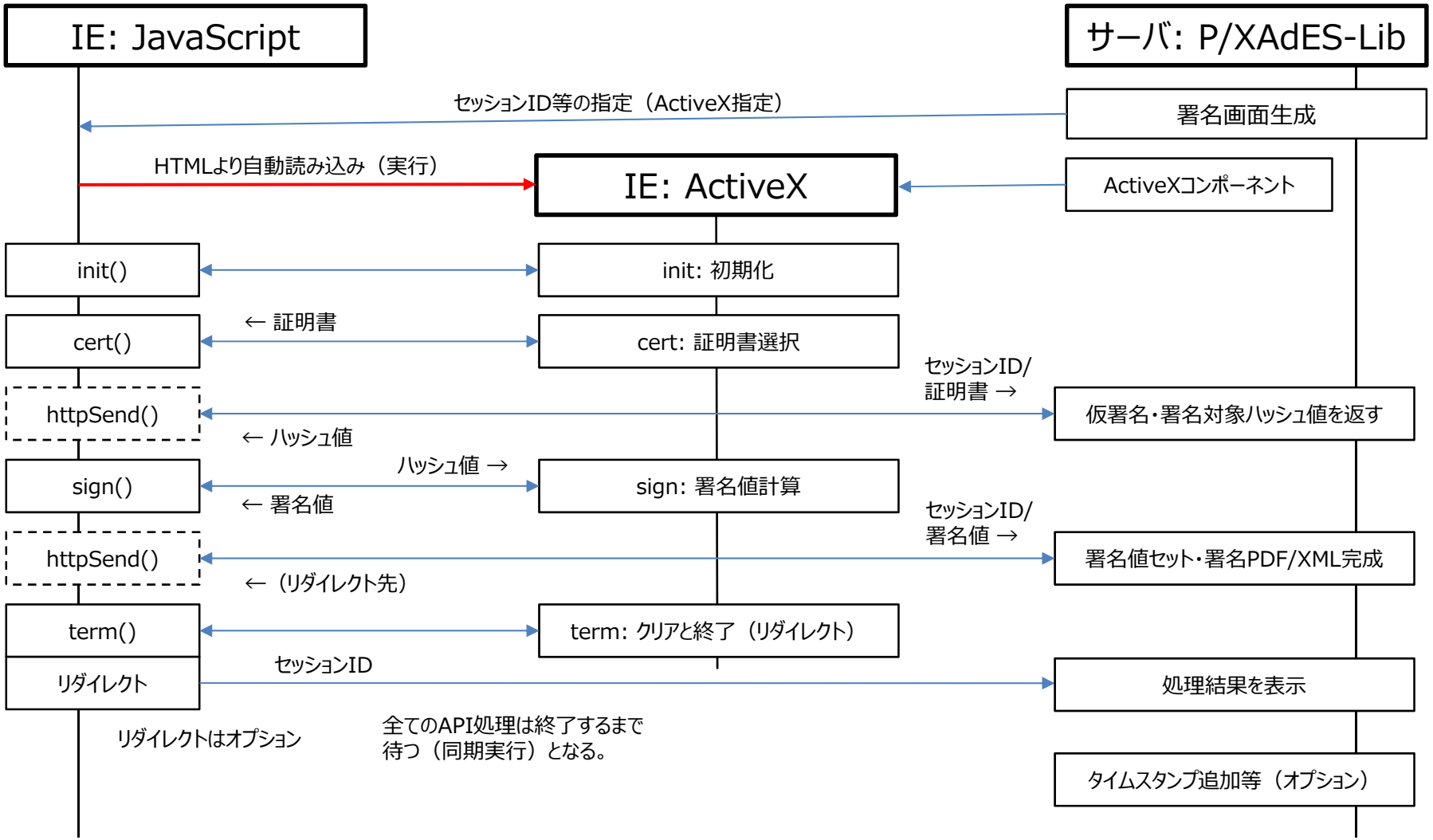
ラング・エッジ クライアント署名の手順

クライアント側 （LE:Client:Sign）	通信 （独自XML）	サーバ側 （LE:PAdES/XAdES:Lib）	サーバ上の ファイル
1. 事前操作と署名実行（ブラウザ）	← 連携 →	0. 署名対象の準備 対象となるファイルの用意 セッションIDの生成（以後通信に利用）	← 情報取得
拡張機能 2. 証明書選択 LE:Client:Sign 証明書の取得/秘密鍵の指定 ＞ 証明書選択画面の表示	← セッションID		署名対象ファイル
	証明書 →	3. 仮署名 LE:PAdES/XAdES:Lib 署名値が無いPAdES/XAdES生成 ＞ 署名証明書が必要 （署名）対象ハッシュ値の計算	
拡張機能 4. 署名値計算 LE:Client:Sign 対象ハッシュ値から署名値を計算 ＞ 秘密鍵の利用	← ハッシュ値		仮署名ファイル
	署名値 →	5. 署名値セット LE:PAdES/XAdES:Lib 署名済みのPAdES/XAdES完成 処理結果（成功/エラー）を返す	
6. 処理結果の確認（ブラウザ） 必要に応じてリダイレクト等で移動 ※ 署名コマンドは直接結果を取得できないのでリダイレクトまたは通信で処理結果を取得する必要がある。	← 処理結果		署名済みファイル
	リダイレクト →	7. 署名後の処理（結果表示等） タイムスタンプ付与や長期署名化も可	→ 保管等

- 仮署名：署名値の埋め込み以外の操作を完了する処理。LE独自の用語。
- 署名値セット：仮署名されたファイルに署名値を埋め込み署名を完了する。
- クライアント・サーバ間の通信はHTTP/HTTPSでXMLを利用した独自手順。
- クライアント側実装はリソースアクセスの為に（C++）ネイティブ実装となる。

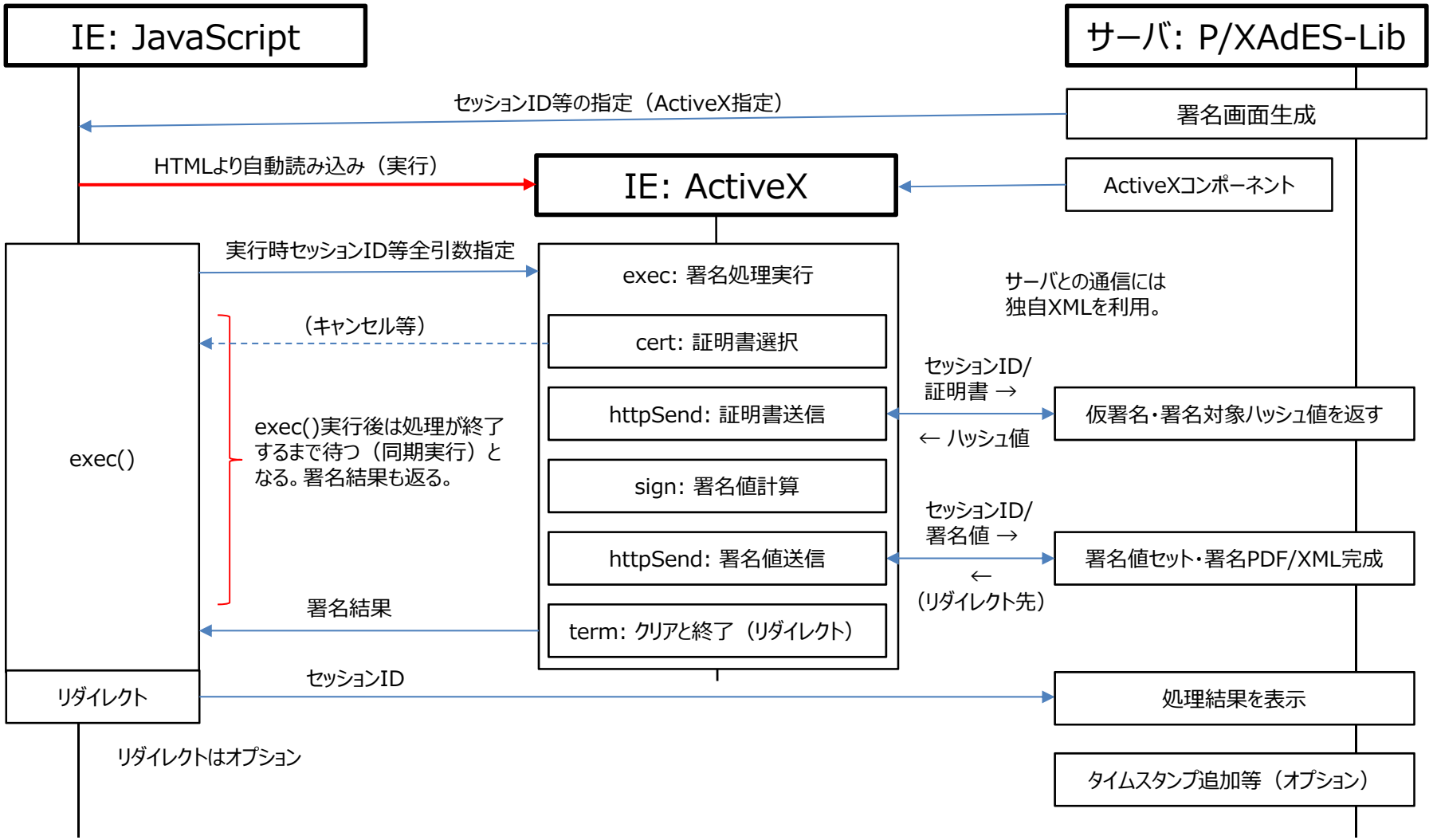
V1/V2: ActiveXコンポーネントのシーケンス図1

ケース1：個別API（cert/sign）を利用した場合



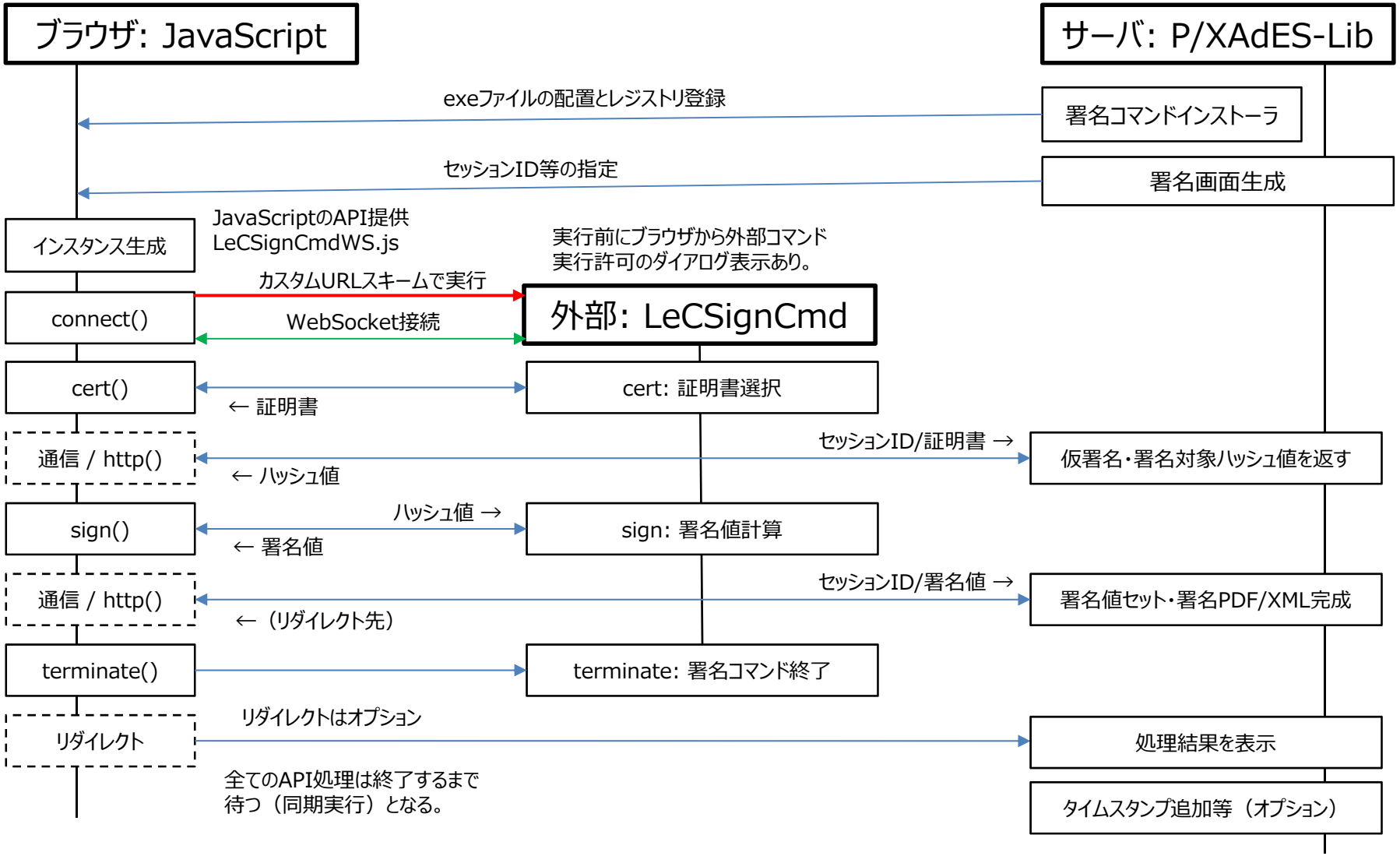
V1/V2: ActiveXコンポーネントのシーケンス図2

ケース2：一括API（exec）を利用した場合



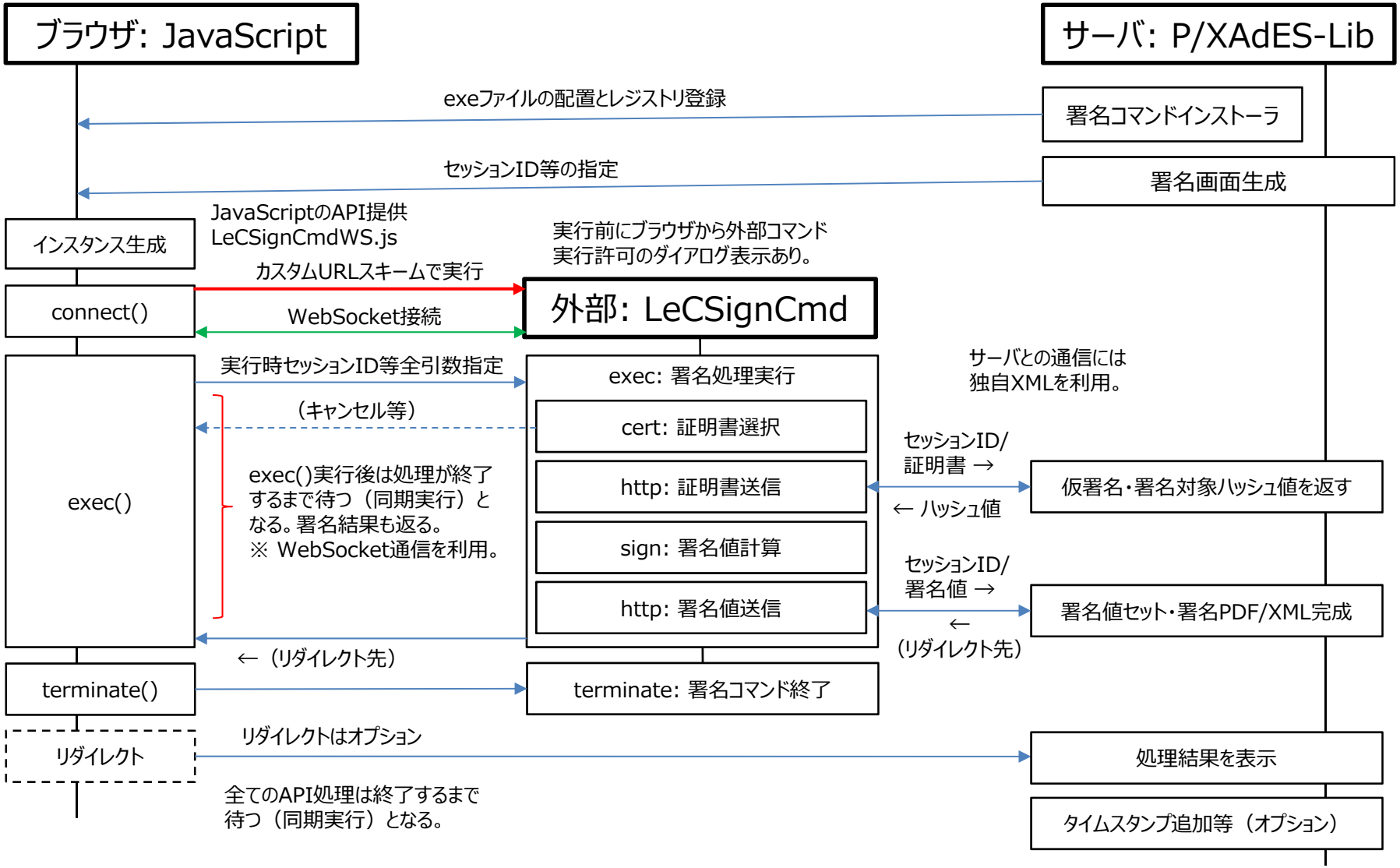
V2のみ: 署名コマンド(カスタムURL)のシーケンス図1

ケース3:個別API(cert/sign)を利用した場合



V2のみ: 署名コマンド(カスタムURL)のシーケンス図2

ケース4:一括API(exec)を利用した場合



クライアント署名 署名コマンド（カスタムURL対応）

実行ファイル名：LeCSignCmd.exe（事前インストールが必要）

その他実行時に必要となるDLL等のファイル：無し（単独実行可）

インストーラ：ファイル配置とレジストリ登録のインストーラを提供

API：JavaScript用API(LeCSignCmdWSクラス)を提供：LeCSignCmdWS.js

```
<html><head>
<!-- LE:署名コマンドJS（※先に指定が必要）-->
<script src="LeCSignCmdWS.js"></script>
<!-- 署名個別実装JS -->
<script type="text/javascript">
  // 署名実行メソッド（引数：セッションID）.
  async function execSign(sessionId){
    var cscmd = new LeCSignCmdWS(initflag, initopt1, initopt2, sid);      // 生成/初期化
    var rc = await cscmd.connect(port, waitSec, cmdSec);                  // 実行と通信接続
    var redirect = await cscmd.exec(cflag, copt, curl, sflag, sopt, …);  // 一括署名実行
    await cscmd.terminate();                                              // 署名コマンド終了
    delete cscmd;                                                         // クリア（削除）
  }
</script>
</head><body>
<input type= “button” value="署名実行" onclick="execSign('session01')">  // 署名実行操作
</body></html>
```

JavaScriptのAPIと署名コマンド間で
WebSocket通信を行うことで同期した
署名コマンドのAPI実行が可能。
※ 署名コマンドが簡易WebScoketサーバになる。

署名コマンド用API: LeCSignCmdWS クラス

JavaScript : LeCSignCmdWS クラス

コンストラクタ: 初期化（初期化と共通する情報のセット）

cscmd = constructor (initflag, initopt1, initopt2, sessionid)

initflag:	署名コマンド 初期化フラグ（CAPI/P11の指定・独自通信XMLプロトコル・ログ記録等の指定）
initopt1/initopt2:	署名コマンド 初期化オプション（P11用DLL指定やログファイルの指定等）
sessionid:	通信に利用するセッションID（execを使うか、独自通信XMLを利用する場合には必須）

署名コマンド接続: 署名コマンドを実行してWebSocket通信を確立する

rc = await cscmd.connect (port, waitSec, cmdSec)

rc:	メソッド戻り値: 正常なら0が、エラーならマイナス値が、返る
port:	接続用のポート番号を指定（0指定時には標準50000番を利用）
waitSec:	JS側接続の待ち秒数を指定（0指定時には標準20秒を利用）
cmdSec:	CMD側接続の待ち秒数を指定（0指定時には標準5秒を利用）

※ 他にも以下のAPIを提供
errorGet : 最後のエラー値を取得
http : HTTP/HTTPS通信の実行
pxml : 独自XML応答の解析

証明書取得: 署名に利用する証明書を選択し取得する（WebSocket通信）

certData = await cscmd.cert (certflag, certopt)

certData:	メソッド戻り値: 正常なら証明書データが、エラーならnullが、返る（後でerrorGetにてエラー値取得可能）
certflag:	証明書取得時のフラグ指定（デフォルト値利用時は 0 を指定）
certopt:	証明書取得時のオプション指定（現在null指定で良い）

署名値計算: 署名対象またはハッシュ値を使って署名値を計算（WebSocket通信）

signData = await cscmd.sign (signflag, signtarget, signalg, signopt)

signData:	メソッド戻り値: 正常なら署名値が、エラーならnullが、返る（後でerrorGetにてエラー値取得可能）
signflag:	署名値計算時のフラグ指定（デフォルト値利用時は 0 を指定）
signtarget:	署名値計算時の署名対象の指定
signalg:	署名値計算時の署名アルゴリズム指定（省略時のデフォルトはRSA-SHA256）
signopt:	署名値計算時のオプション指定（PKCS#11利用時にPIN指定可能）

一括実行: 証明書取得（cert）/署名値計算（sign）とサーバ通信（http）を一括実行する（WebSocket通信）

redirect = await cscmd.exec (cflag, copt, curl, sflag, sopt, signAlg, surl, head, hflag)

署名コマンド終了: 署名コマンドを終了してWebSocket通信を開放する（実行しないと署名コマンドが終了しない）

await cscmd.terminate ()

参考：カスタムURL(カスタムURI)スキームの登録

レジストリに設定してURL(プロトコル)とアプリを関連付ける。

※ 仕組みとしては拡張子関連付けに近い。

レジストリ登録 SignCmd.exe を lcsign: に 関連付け。	[HKEY_CLASSES_ROOT¥lcsign01] @="URL:LE Client Sign Protocol" "URL Protocol"="" [HKEY_CLASSES_ROOT¥lcsign01¥DefaultIcon] [HKEY_CLASSES_ROOT¥lcsign01¥shell¥open¥command] @="¥"C:¥¥Program Files¥¥LE¥¥SignCmd.exe¥" ¥"%1¥""
HTML記述 引数 sid 指定	<HTML><BODY> LE Sign </BODY></HTML>

URL引数を渡す

呼び出される署名コマンドを事前にインストールしておく必要がある。
レジストリ関連付けをインストーラで事前に登録しておく必要がある。

インストーラが別途必要

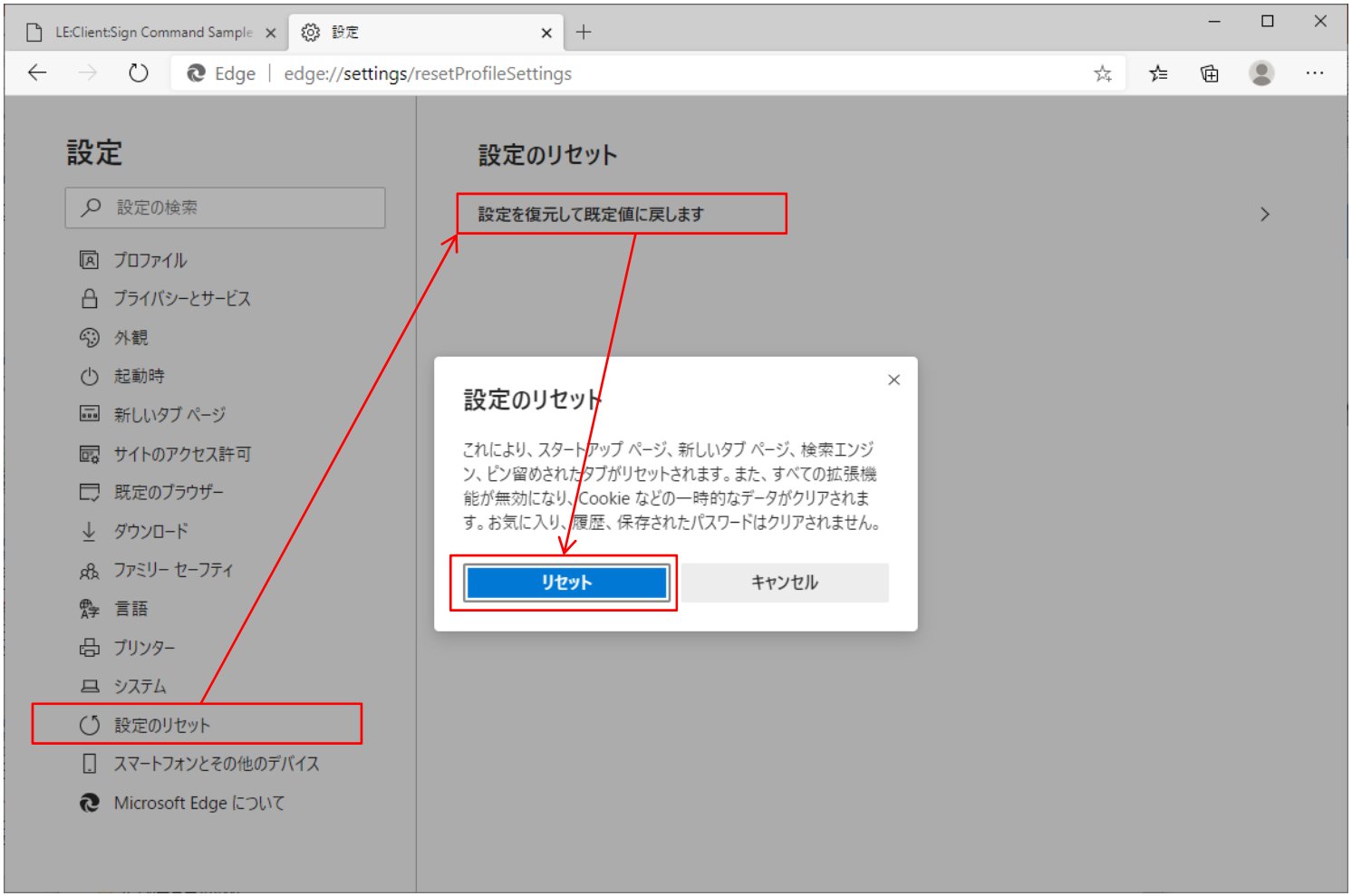
HTMLを動的生成することで引数としてセッションIDや署名対象ハッシュ値を渡せる。
コマンド実行中に最前面ウィンドウ（Chrome等）を親としたモーダルウィンドウを開く。

※ 試験用に署名コマンドでレジストリ登録と削除が可能。

Regist.bat：同じフォルダにある署名コマンドをレジストリ登録 / Unregist.bat ： レジストリ削除

「常に許可済み」の解除方法1（設定画面からリセット）

「常に許可済み」の設定情報はブラウザ（Edge/Chrome）が持つ。
設定画面から解除するには「**設定のリセット**」を行うしかない。



「常に許可済み」の解除方法2（設定ファイルの編集）

「常に許可済み」の設定情報はブラウザ（Edge/Chrome）が持つ。
個別に解除するには「**設定ファイルの編集**」を行うしかない。

場所：
C:¥Users¥[ユーザ名]¥AppData¥Local¥Microsoft¥Edge¥User Data¥Default

ファイル名：
Preferences

変更：カスタムURLスキーム名を検索してフラグをfalseに変更

{"allowed_origin_protocol_pairs":{"file://":{"lcsign01":**true**}}}
↓以下に変更
{"allowed_origin_protocol_pairs":{"file://":{"lcsign01":**false**}}}

※ file:// の部分はサイト名となる。

「常に許可」を使う場合には、個別解除が難しいので注意が必要。

LE:Client:Sign V2.1 新機能

1. XAdES-BES署名機能（サーバ連携不要）

- V2.0まではXAdES署名の生成は、サーバ連携が必須であったが、V2.1では新たにXAdES-BES署名データを生成する新APIの xades() が追加された。
- 新APIの xades() ではタイムスタンプは付与できないが、サーバ側のLE:XAdES:Libを使ってXAdES-T（タイムスタンプ付）やXAdES-A（長期署名）にすることが可能。
- 利用方法は、署名APIの sign() の代わりに xades() を使う。署名対象を増やす場合には新APIの xadd() を利用する。Enveloped/Detahced/Envelopingに対応。
- 用途：クライアント側でXAdES署名を行いサーバに送ることが出来る。

2. 証明書情報の取得（サーバ連携不要）

- 既存の証明書取得APIの cert() で取得したデータから、証明書の種類や情報を取得する新APIの info() が追加された。形式はJSONかXMLで取得可能。
- 判別可能：マイナンバーカード署名証明書・商業登記電子証明書・それ以外
- 共通取得：種別・発行者名・所有者名・シリアル番号・有効期間（開始と終了）
- 個別取得：マイナンバーカード4情報と、商業登記4情報他
  - マイナンバーカード4情報：氏名・住所・生年月日・性別
  - 商業登記4情報他：法人名・本店住所・代表氏名・代表肩書・会社法人等番号（12桁）
- 用途：署名前に証明書種別や有効期間のチェックが出来る。

LE:Client:Sign V2.1 証明書情報サンプル

商業登記証明書の例（JSON形式）

```
{
  "type":"REGIST",
  "serial":"072C75FXXXXXXX",
  "from":"20XX-12-13T10:38:31",
  "to":"20XX-03-13T23:59:59",
  "issuer":{
    "c":"JP",
    "o":"Japanese Government",
    "ou":"Ministry of Justice",
    "cn":"Registrar of Tokyo Legal Affairs Bureau"
  },
  "subject":{
    "c":"JP",
    "o":"MOJ No.0100XXXXXXXX",
    "cn":"040101XXXXXXX-kaisataro"
  },
  "regist":{
    "number":"0100XXXXXXXX",
    "issue":"東京法務局",
    "corp":"株式会社〇〇〇〇",
    "address":"東京都千代田区〇〇町三丁目１番地１号",
    "name":"会社太郎",
    "title":"代表取締役"
  }
}
```

マイナンバーカード証明書の例（XML形式）

```
<CERT>
  <type>JPKI</type>
  <serial>703XXX</serial>
  <from>20XX-03-15T03:38:08</from>
  <to>20XX-04-01T23:59:59</to>
  <issuer>
    <c>JP</c>
    <o>JPKI</o>
    <ou>JPKI for digital signature</ou>
    <ou>Japan Agency for Local Authority Information Systems</ou>
  </issuer>
  <subject>
    <c>JP</c>
    <l>Tokyo-to</l>
    <l>Edogawa-ku</l>
    <cn>20XXXXXXXXXXXXXXXXXXXXXX03A</cn>
  </subject>
  <jpki>
    <name>公的　太郎</name>
    <scname>00000</scname>
    <address>東京都江戸川区〇〇町１丁目２３番４５号</address>
    <scaddress>0000000000000000000</scaddress>
    <birth>319810401</birth>
    <gender>1</gender>
  </jpki>
</CERT>
```