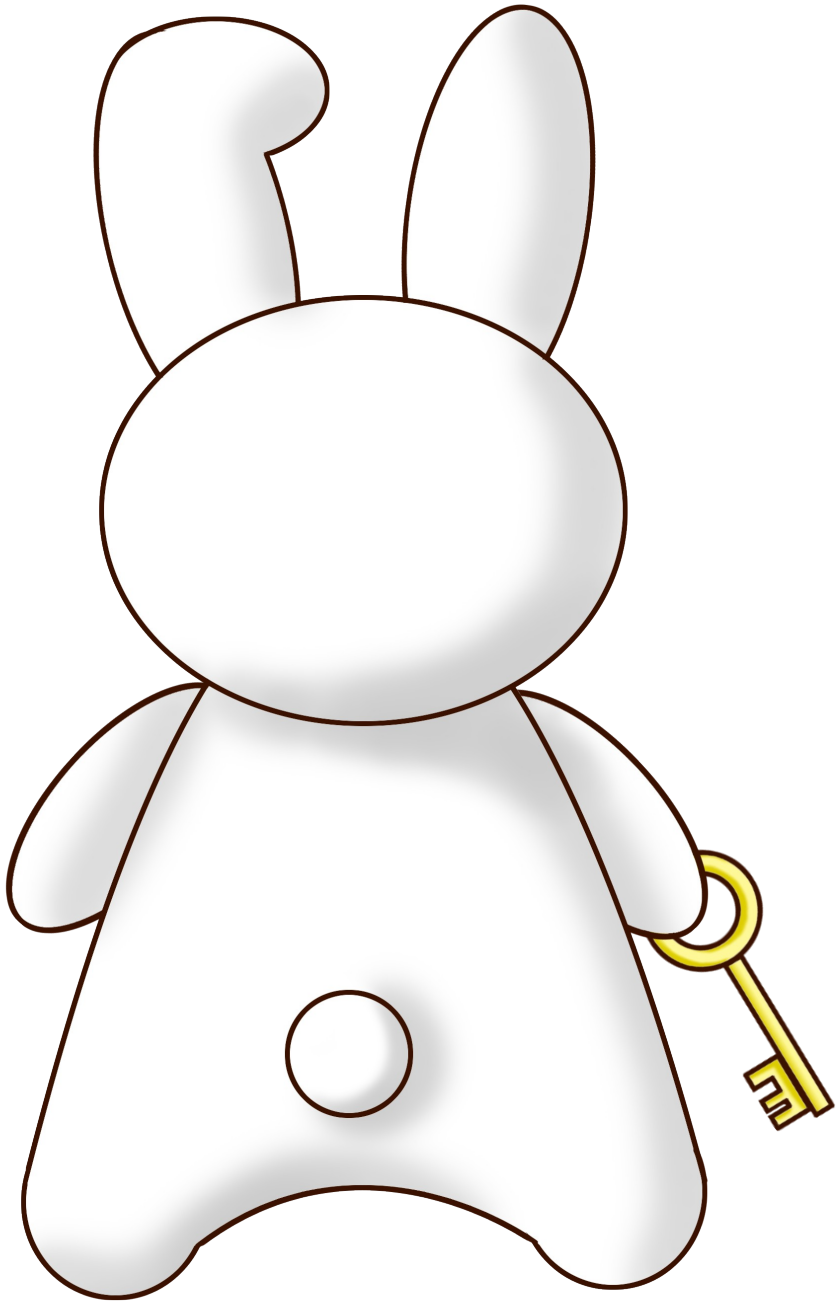
マイナンバーカードと電子署名の本



@hamano

第２版

マイナンバーカードと電子署名の本

濱野　司

はじめに

マイナンバーカードというとどんな印象があるでしょうか?

「使いみちが無い」「なんとなく怖い」

一般的にその様な印象を持っている方が多いようです。マイナンバー（個人番号）に関して様々な議論がありますが、この本は個人番号制度とは直接関係ありませんし触れないようにしています。

これは自分のマイナンバーカードに搭載されている IC チップにアクセスして遊ぼう、という趣旨の本です。具体的には OpenSC というオープンソース・ソフトウェアを使いながら公的個人認証の仕組みとその将来性について解説します。

公的個人認証は私達のオンライン生活をより安全・便利にする可能性を持っています。しかし、このサービスが広く活用されるにはまだ多くの課題があります。

まず国民全体に対する認知度が足りていません。とりわけ、この本の対象読者である技術者にさえよく知られていないシステムが、安心して利用できる社会基盤として世の中に普及することは難しいでしょう。

1 章はマイナンバーカードと公的個人認証の技術的な背景を説明します。総務省はマイナンバーカードの普及に取り組んでいるようですが、何故かカードの技術仕様をあまり公開したがりません。2 章では OpenSC というソフトウェアを使いながらマイナンバーカードにアクセスする方法を紹介します。実際に手を動かしながらマイナンバーカードで暗号・署名の操作をやってみましょう。

残念ながら日本の公共システムの多くは Windows 限定・IE 限定といった限られたプラットホームでしか動作しません。OpenSC はクロスプラットホームなソフトウェアですので Windows、macOS、Linux、FreeBSD など様々な環境をサポートしています。

本書ではマイナンバーカードのブラックボックスを明らかにし、漠然とした不安を取り除ければ幸いです。

目次

第 1 章
マイナンバーカード

マイナンバーカードはいくつかの役割を持っています。表面には氏名・住所・顔写真が記載されていて、これまで運転免許証で行っていたような本人確認に利用できます。また、裏面にはマイナンバー (個人番号) が記載されていて、税金や年金、社会保障のために利用されます。これらはカードという物理的な特徴を用いた機能ですが、カードに搭載されている IC チップは更にいくつかの機能を持っています。

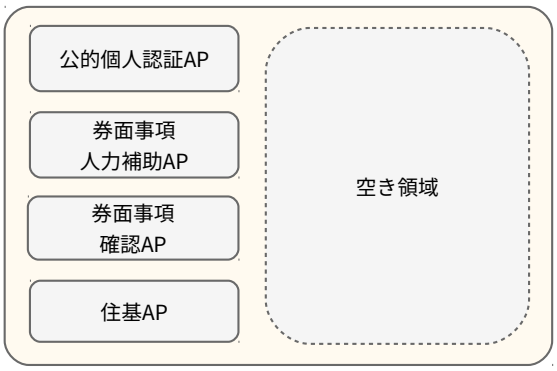


図 1.1　マイナンバーカードの IC チップ

マイナンバーカードの IC チップの中はこんな風になっているよ、と総務省は説明しています。上から、公的個人認証 AP、券面事項補助入力 AP、券面事項確認 AP、住基 AP、と 4 つの機能が入っていて、空き領域には新しい機能を追加できます。

しかし、私たち技術者はこの様なふんわりした説明では満足できません。カードにどの様なデータが格納されていて、どの様にアクセスするのかを知りたいはずです。

1.1 公的個人認証とは

公的個人認証は 2004 年、住民基本台帳カード (以降住基カード) の頃にサービスを開始した PKI をベースとした本人確認サービスです。当初は行政手続きをオンラインで行うための用途に限られていました。当時私は 500 円を握りしめワクワクしながら証明書を発行して貰ったものですが、ほとんどの行政手続きは Linux デスクトップに対応していませんでした。

なんとか Linux で使いたいという思いでリバース・エンジニアリングを試みましたが、特に使い道が無い事が判り、結局住基カードはタンスの肥やしとなってしまうのでした。

マイナンバーカードでは用途が拡大されて民間サービスでも利用できるようになりました。また、マイナンバーカードにはデジタル署名用の証明書に加え、ユーザー認証用の証明書が追加されました。デジタル署名用証明書は申込書や契約書などの文書へのデジタル署名に利用します。ユーザー認証用証明書は Web サイトなどへのログインで利用します。

この 2 種類の証明書については後ほど詳しく説明していきます。

1.2 IC チップの特徴

マイナンバーカードには 4 つの AP が入っていると説明しました。AP とはアプリケーション・プログラムの略です。マイナンバーカードの IC チップは SD カードのようなただのストレージではありません。カード内で独立して駆動するプロセッサーがカード OS の上でアプリケーションを実行します。各 AP はカード OS 上で独立しており、お互いに干渉しません。

マイナンバーカードは共通鍵暗号 (AES)、公開鍵暗号 (RSA) 機能を持っています。この RSA 秘密鍵は IC チップのセキュアエレメントに格納されていて暗号化・署名処理は IC チップの中で安全に行います。ですから仮に利用端末が危殆化したとしても秘密鍵を盗まれることはありません。

また、マイナンバーカードは耐タンパー性を備えています。暗号化の処理時間や消費電力を計測して秘密鍵を推測するサイドチャネル攻撃や、チップをスライスして電子顕微鏡で覗くといった攻撃に対して耐性を持っています。

マイナンバーカードの IC チップは接触型・非接触型の 2 種類のインターフェースを持っています。接触型であれば ISO/IEC 7816 に対応したカードリーダー、非接触の場合 NFC カードリーダーでアクセスできます。

1.3 NFC 規格

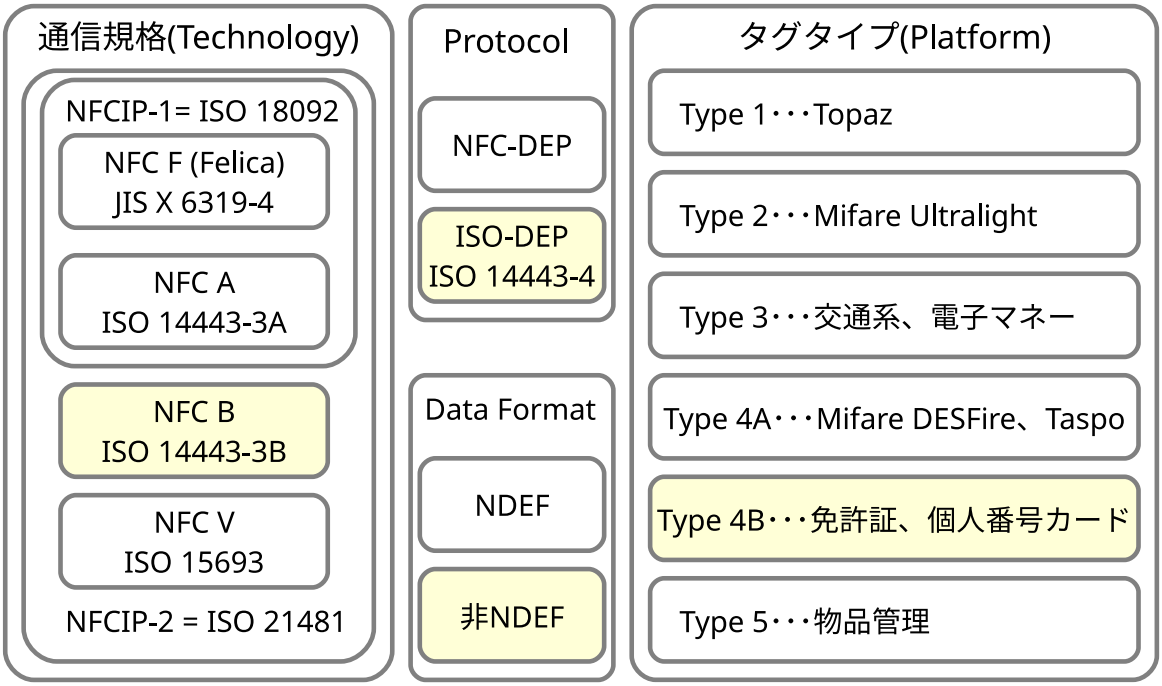


図 1.2　マイナンバーカードに必要な NFC 規格

NFC 仕様は様々な国際規格の集合体です。ここで個別の規格の詳細は説明しませんが、NFC でマイナンバーカードにアクセスするためには NFC-B の通信規格と Type 4B タグ仕様、ISO-DEP、非 NEDF フォーマットをサポートする必要があります。

国内で出回っている Android 端末の多くは NFC リーダーを搭載しており、ほとんどの端末でマイナンバーカードにアクセスできます。しかし、いくつかの端末は NFC F の FeliCa のみをサポートし NFC B を利用できない場合があるので注意してください。

そして iOS11 の iPhone でも NFC 対応が始まりましたがまだマイナンバーカードにアクセスすることはできませんでした。iOS11 および iOS12 は ISO-DEP をサポートしていなかったからです。しかし、2019 年後半にリリースされる iOS13 から ISO-DEP がサポートされ、晴れて iPhone でマイナンバーカードを活用できる環境が整ってきました。

1.4 データモデル

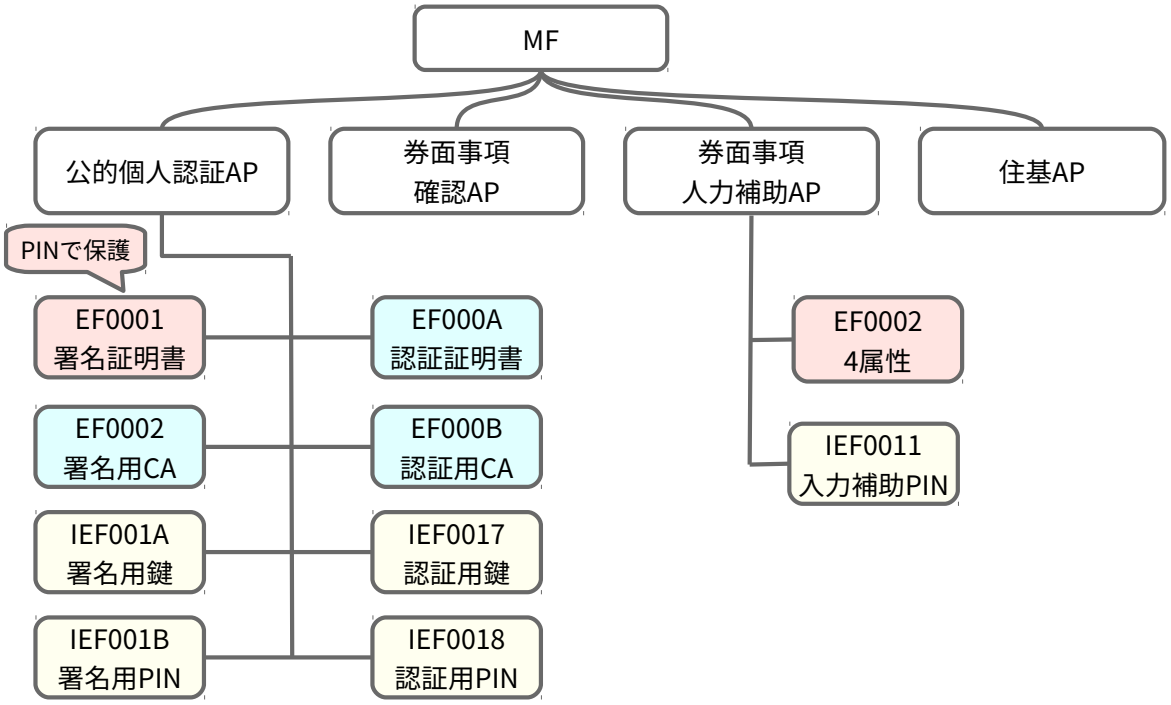


図 1.3　マイナンバーカードのデータモデル

これは筆者が調査したマイナンバーカードのデータモデルです。

IC カードのデータは私達が普段利用しているファイルシステムのようなディレクトリ構造になっています。そしてファイルシステムと同様にファイルに対して open や read といった操作を行います。具体的には ISO/IEC 7816-4 で定義される APDU(Application Protocol Data Unit) というバイナリプロトコルでやり取りします。

公的個人認証 AP には、証明書、CA 証明書、秘密鍵が 2 セットあります。これは前述したユーザー認証用とデジタル署名用の 2 つに対応します。

1.5 デジタル署名

公的個人認証 AP には RSA 2048bit 鍵とそれに対応する X.509 証明書が入っています。これらを用いてデジタル署名およびユーザー認証処理を行うことが出来ます。デジタル署名用証明書は以前から住基カードに格納されており、e-Tax などの行政手続きで使われてきました。

公的個人認証のデジタル署名用証明書は、住民票に登録されている氏名・住所・年齢・性別 (以下 4 属性) を含んでいます。具体的には X.509 Subject Alternative Names にこれらの 4 属性が記載されています。

これらの情報は住民票に基づいているので正確な本人確認を行うことができます。つまり、これまで住民票のコピーを提出しろと言われるような紙の手続きをオンライン経由のデジタル署名によって代替することができます。

このデジタル署名は印鑑による押印を代替することもできます。マイナンバーカードの中にはあなたの実印が入っていると言えます。

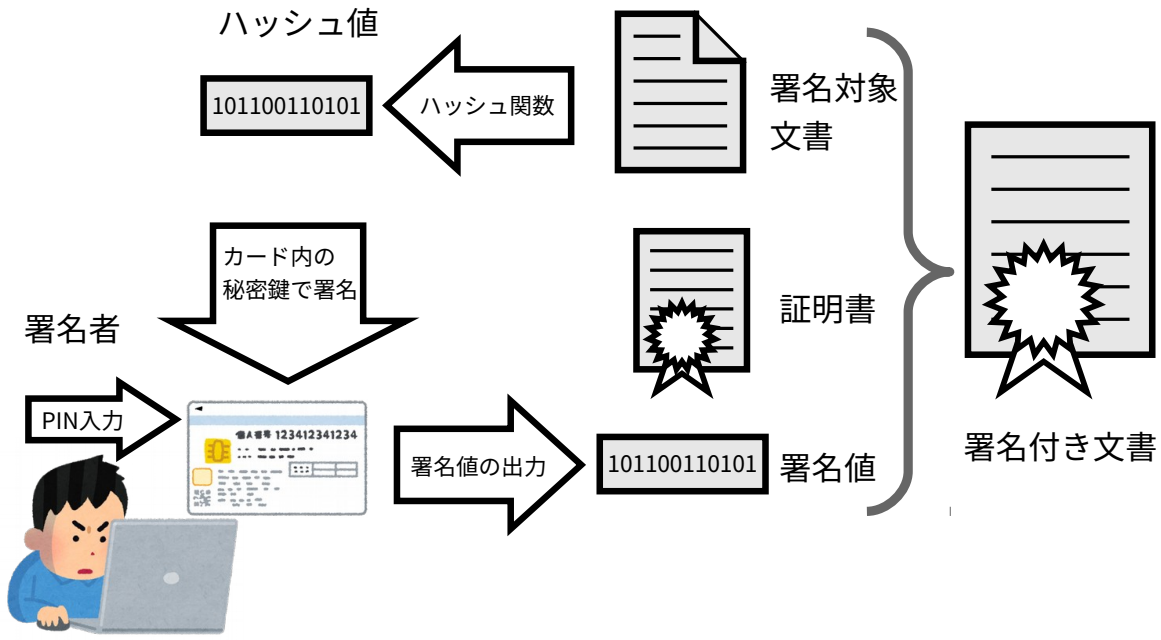


図 1.4　デジタル署名

1.6 ユーザー認証

マイナンバーカードにはユーザー認証用の証明書と秘密鍵も格納されています。これは従来のユーザー名 + パスワードによる認証を代替するものです。現時点でこれを利用しているのはマイナポータルや住民票の交付サービスくらいですが、将来的には銀行のオンラインバンキングなど民間サービスでの活用も想定されています。

従来のシステムで長らく使われてきたユーザー名 + パスワードによる認証方式は多くの問題と誤解を抱えています。利用者側には長いパスワードを覚えられない・覚えたくないという問題があり、サービス提供者側の問題としてはパスワード文字種の組み合わせを強制したりパスワードの定期変更を強制するといった誤ったセキュリティポリシーが蔓延しています。

普段から公開鍵認証方式を用いてサーバーに SSH ログインしている技術者には説明不要と思いますが、ほとんどの場合パスワード認証より公開鍵認証の方が安全です。[*1]

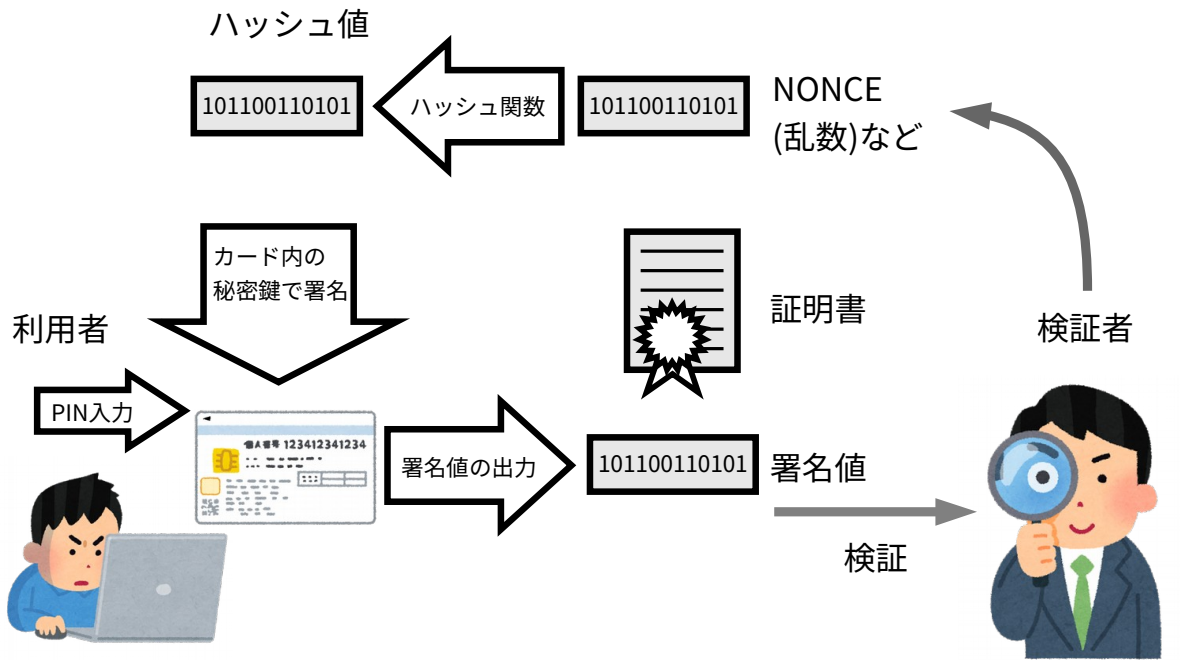


図 1.5　ユーザー認証

[*1] パスワード認証と公開鍵認証について、それぞれに長所と短所があり直接比較できるものでは無いが、マイナンバーカードを用いる場合、秘密鍵の利用にローカル認証 (PIN) を求めている事と、物理的に秘密鍵を複製できない事により、NIST SP 800-63 で定義される記憶 + 所有 (ハードウェア) AAL3 を満たす。したがってパスワード単独による認証 (AAL1) より安全な認証方式と言える。

1.7 デジタル署名とユーザー認証

公的個人認証では 2 種類の証明書を利用します。デジタル署名とユーザー認証について、実際に行われている処理の内容にさほど違いありません。しかし、利用者はデジタル署名とユーザー認証を異なる行為として区別しなければなりません。なぜこれら 2 つの証明書を使い分ける必要があるのでしょうか。

X.509 証明書には証明書の用途 (Key Usage) という項目があります。デジタル署名用証明書の `Key Usage` には `Non Repudiation` というフラグが有効になっています。これは否認防止というフラグで、署名した文書への同意を意味します。ですから責任能力のない 15 歳未満の子供にはデジタル署名用証明書は発行されない制度になっています。

一方ユーザー認証用証明書に `Non Repudiation` フラグはありません。この証明書の署名対象は NONCE などの乱数であって同意したい文書ではないからです。もし仮に、これらの区別を行わずデジタル署名用証明書でユーザー認証を行うとどうなるでしょうか。悪意のある Relying Party が NONCE の代わりに契約文書を送ってきて利用者の意図しない電子契約を行うかもしれません。

1.8 印鑑と電子署名

印鑑は古代メソポタミア文明の頃に発明されたとされています。当時は円柱状の印鑑を粘土板の上で転がしながら押し付けるという方法で押印が行われました。現代において、この印鑑を押す習慣が残った数少ない国のひとつが日本です。銀行口座を作るとき、家を借りるとき、重要な契約を取り交わすシーンで印鑑が必要になります。私たちはなぜ印鑑を押すのでしょうか。

日本には印鑑登録という制度があるものの、印鑑を押すこと自体に法律的な効力があるわけではありません。あくまで裁判となった場合に本人が文書に同意したと推定する材料として利用されます。

今更この本の読者にデジタル署名と印鑑のどちらが安全かという説明は必要ないでしょう。3D プリンタが発達した昨今、そこらへんの子供でも印鑑の偽造が可能です。もはや印影はただの装飾であり押印は形骸化した儀式でしかありません。私たちはいつまでこのレガシーを引きずるのか、という課題を突きつけられています。

電子署名の普及が進まなければ、私たちの社会はある種の痛みを伴うでしょう。印鑑の偽造による詐欺事件や、官僚による公文書の偽造も私たち全員が受けた痛みのひとつです。

1.9 エストニアの事情

エストニアは世界でもっとも電子行政サービスが充実した国としてよく取りざたされます。日本で公的個人認証が始まる以前に、エストニアでは国民に eID カードと呼ばれるカードが配布されました。この eID カードには日本の公的個人認証と同じく 2 種類の電子証明書が格納されていて、電子認証・電子署名を活用した行政・民間サービスが普及しています。ここではこの eID カードと日本のマイナンバーカードと比較して、エストニアならではの電子サービスをいくつか紹介します。

電子メールアドレス

eID カード保有者は、生涯不変の転送メールアドレスを利用できます。メールアドレスは<名>.<氏>_<4 桁の乱数>@eesti.ee という形式で、eID カード使って転送先メールアドレスを設定します。

このメールアドレスは基本的に行政機関から国民への連絡に利用しますが、個人的に利用することもできます。

電子投票

エストニアは世界で最初にインターネット選挙を実施した国です。インターネット選挙が実現すれば投票所から遠い場所に住む有権者の負担が少なくなる反面、安全性が懸念されるでしょう。エストニアの電子投票では、eID カードを使った電子認証により 1 人 1 票という原則を守り、電子署名により投票の意思を示します。そして投票の匿名性という民主主義に必要な要件も同時に満たしています。

モバイル ID

eID カードの欠点はカードリーダーが無いと利用できないという点です。モバイル ID は携帯電話の SIM カードに電子証明書を格納することで携帯電話だけで電子認証と電子署名を行えるようになるサービスです。これは日本でも近いうちに同様のサービスが開始されるでしょう。

第 2 章
OpenSC であそぼう

マイナンバーカードと OpenSC で遊ぶにあたり、注意して頂きたいことは法令の遵守です。マイナンバーカードに関する法令は幾つかありますが特に厳しいのはマイナンバー (個人番号) に関する法令です。マイナンバー (個人番号) を目的外 (社会保障・税務・災害対策) で利用することは厳しく制限されています。

また、他人の公的個人認証の証明書を収集・記録することも制限されています。これを行うには総務大臣の認可が必要になります。

要点をまとめると、他人のマイナンバーカードからデータを読み出して記録しないようにしましょう。これだけ注意すれば、うっかり法律に違反してしまうことは無いでしょう。

自分のマイナンバーカードのデータを読み出したり、アクセスを禁止する法律はありません。もしそんな法律があれば誰もマイナンバーカードを利用できなくなってしまいます。必要以上に怖がらず正しく遊びましょう。

2.1 OpenSC とは

OpenSC はクロスプラットフォームで動作するスマートカード用のツール・ライブラリ群です。主にスマートカードとアプリケーションを繋ぐミドルウェアとしての役割が主体です。OpenSC は Web ブラウザやメーラーなどのアプリケーションで利用することもできますが、OpenSSL や NSS などの暗号化ライブラリのバックエンドとして利用することも可能です。

スマートカードは標準仕様に従ったものもあれば、カードの用途やベンダー固有の独自仕様だったりと様々です。OpenSC はカード固有の仕様を吸収し、標準仕様に適合させるために各種カードドライバを提供しています。

カードドライバは米国の公務員が身分証として利用している PIV カードやベルギーの国民 ID カード Belpic などがサポートされています。筆者はこれらに加えマイナンバーカードの公的個人認証を OpenSC で利用する為のカードドライバを開発しました。これにより公的個人認証の PKCS#11 API や PKCS#15 エミュレーションを実現しています。

OpenSC はクロスプラットホーム対応していますので、Windows, macOS, Linux など多くの動作環境で動作します。

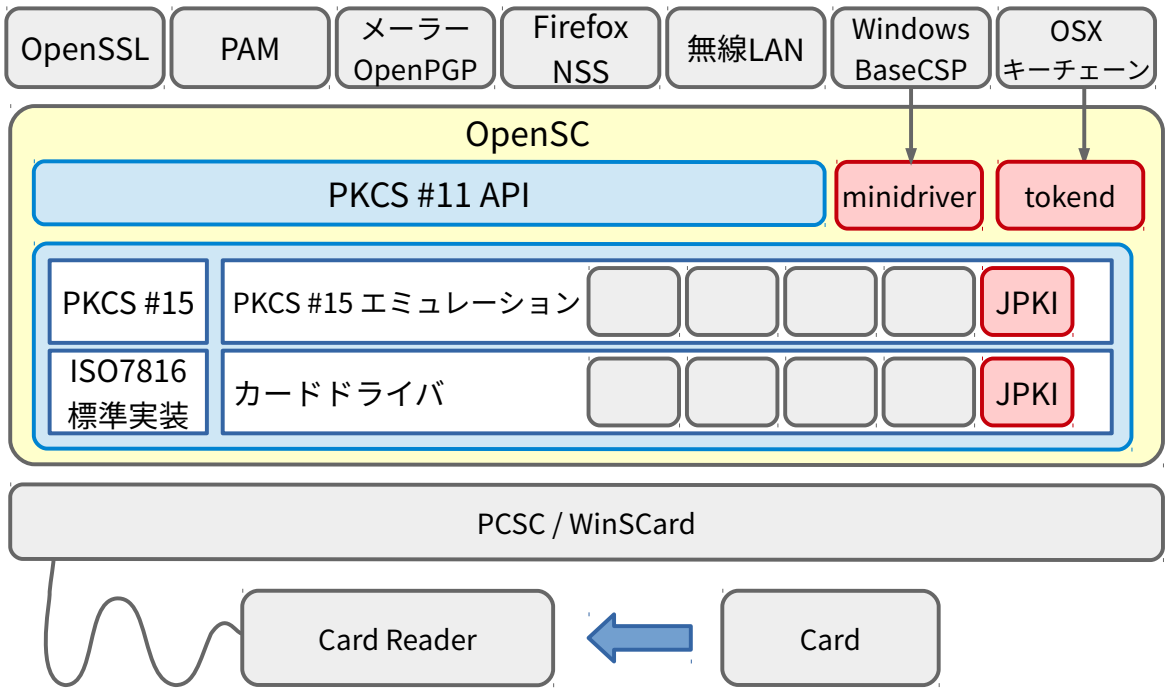


図 2.1　OpenSC アーキテクチャ

2.2 OpenSC のインストール

OpenSC が公的個人認証に対応したのはバージョン 0.17.0 以降となります。古い OS の標準パッケージを利用すると公的個人認証を利用できない場合があるので注意してください。(例: Debian Stretch など)

Debian Buster, Ubuntu 18.04 LTS では以下のようにして公的個人認証に対応した OpenSC をインストールできます。

```
$ sudo apt install -y opensc
```

Windows, macOS の人は以下のページから最新リリース版のインストーラーをダウンロードしてください。

https://github.com/OpenSC/OpenSC/wiki

2.3 マイナンバーカードの暗証番号

役所でマイナンバーカードを受け取った時、2 種類または 4 種類の暗証番号を設定したはずです。それぞれ以下の通りです。

a) 公的個人認証のデジタル署名用パスワード (英数字 6-16 文字)
b) 公的個人認証のユーザー認証用暗証番号 (4 桁数字)
c) 住基 AP 用暗証番号 (4 桁数字)
d) 券面入力補助 AP 用暗証番号 (4 桁数字)

(b) と (c) と (d) は同じ 4 桁の数字ですので、同じものを設定させる自治体が多いようです。今回 OpenSC で利用するのは (a) と (b) の公的個人認証用のパスワードと暗証番号です。デジタル署名用パスワードは 5 回、ユーザー認証用暗証番号は 3 回間違えるとブロックされ、それ以上入力できなくなります。こうなってしまったら役所でリセットするしかありませんので暗証番号は注意して入力してください。

2.4 pkcs15-tool の使い方

ここではマイナンバーカードを使って OpenSC に付属するツールの使い方を説明します。pkcs15-tool は証明書や公開鍵など、カードの内部データにアクセスするためのコマンドラインツールです。まずカード内のすべての PKCS#15 オブジェクトをダンプします。PIN オブジェクト、秘密鍵オブジェクト、公開鍵オブジェクト、証明書オブジェクトをそれぞれ 2 セット確認できます。

```
$ pkcs15-tool -D
```

暗証番号やパスワードを間違えてしまった時、残り何回間違えても大丈夫なのか知りたくなります。pkcs15-tool の`--list-pins` オプションは PIN がブロックされるまでの残り回数を確認できます。

```
$ pkcs15-tool --list-pins
```

ユーザー認証用証明書を表示します。ユーザー認証用証明書は証明書番号 1 に対応付けされています。証明書は PEM 形式で出力されますので openssl コマンドでテキスト形式に変換して表示します。

```
$ pkcs15-tool -r 1 | openssl x509 -text -noout
```

証明書 ID は以下のように対応しています。

1. ユーザー認証用証明書
2. デジタル署名用証明書
3. ユーザー認証用 CA 証明書
4. デジタル署名用 CA 証明書

続いて、デジタル署名用証明書を表示します。これは証明書番号 2 に対応付けされていますが、このままでは出力できません。デジタル署名用証明書はパスワードで保護されているので、証明書を参照するにはパスワードを入力する必要があります。`-a` オプションで PIN オブジェクト 2 を指定し、公的個人認証のデジタル署名用パスワードを入力します。(パスワードのアルファベットは大文字で入力してください。)

```
$ pkcs15-tool -r 2 -a 2 --verify-pin | openssl x509 -text -noout
```

openssl コマンドに慣れた方は、ユーザー認証用証明書を CA 証明書で検証する動作を確認してみても良いでしょう。

```
# ユーザー認証用証明書を出力
$ pkcs15-tool -r 1 > auth.pem
# ユーザー認証用 CA 証明書を出力
$ pkcs15-tool -r 3 > authCA.pem
# CA 証明書で証明書を検証
$ openssl verify -CAfile authCA.pem auth.pem
auth.pem: OK
```

2.5 pkcs15-crypt の使い方

pkcs15-crypt は IC カードでデジタル署名を行うためのコマンドラインツールです。このツールを使って公的個人認証のデジタル署名を試してみましょう。

まず署名対象のメッセージを作成します。今回は hello という内容のテキストファイルに署名を行います。

```
$ echo "hello" > message.txt
$ pkcs15-crypt -s -k 2 --pkcs1 -R -i message.txt -o message.signed
```

--pkcs1 オプションは署名対象の 6byte のテキストを鍵長の 2048bit にパディングします。-k 2 オプションを指定してデジタル署名用の鍵を指定していますのでここではデジタル署名用のパスワードを入力してください。これで署名完了です。

続いてこの message.signed を署名検証してみましょう。まず、署名用証明書から公開鍵を取り出してこの公開鍵で署名検証します。

```
# デジタル署名用証明書から公開鍵を取り出す
$ pkcs15-tool -r 2 | openssl x509 -noout -pubkey > sign.pub
# 公開鍵で署名検証
$ openssl rsautl -verify -pubin -inkey sign.pub -in message.signed
hello
```

元のメッセージである hello が出力されました、これにより message.signed は秘密鍵の所有者が署名したファイルであることが確認できます。

今回は、小さいテキストファイルに対して署名を行いましたが、対象ファイルが 256byte(2048bit) を超えると RSA 署名できません。ですので通常デジタル署名は署名対象のダイジェストに対して署名を行います。

2.6 Python で PKCS#11 ライブラリを使う

PKCS#11[*1]はスマートカードなどの暗号デバイスを利用する為の API です。OpenSC が提供している PKCS#11 ライブラリを利用して C、Java、Python など各種プログラミング言語で IC カードにアクセスできます。標準化された PKCS#11 API を利用することで開発者は特定の暗号デバイスに依存しないアプリケーションを実装できます。

PKCS#11 ライブラリの大まかな利用方法は以下のとおりです。

1. ライブラリをロードして初期化 - `C_Initialize()`
2. スロットを取得 - `C_GetSlotList()`
3. セッションを開く - `C_OpenSession()`
4. PIN を入力してログインする - `C_Login()`
5. 証明書の取得や暗号化、署名処理などを行う
6. ログアウトする - `C_Logout()`
7. セッションを閉じる - `C_CloseSession()`
8. ライブラリの終了処理 - `C_Finalize()`

ここでは Python スクリプトで OpenSC の PKCS#11 ライブラリを利用してデジタル署名を行う方法を紹介します。まず Python の PKCS#11 ラッパーモジュール `PyKCS11` をインストールします。

```
$ pip install PyKCS11
```

2.6.1 ライブラリをロード

PyKCS11 は環境変数 `PYKCS11LIB` に指定した共有ライブラリをロードします。ライブラリのパスは Debian 系 OS では`/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so` です。Windows では DLL へのパスに置き換えてください。

以下のスクリプトで PKCS#11 ライブラリをロードします。

```
from PyKCS11 import *
pkcs11 = PyKCS11Lib()
pkcs11.load()
```

[*1] もともとは RSA 社が標準化した古い仕様ですが OASIS に移行した後、現在でも活発に改良が提案されています。

2.6.2 スロットを取得

PKCS#11 はスロットという単位で証明書・秘密鍵を論理的にグループ化しています。OpenSC では公的個人認証のユーザー認証用をスロット 0、デジタル署名用をスロット 1 に割り当てています。

公的個人認証のユーザー認証用のスロットは以下のように取得します。

```
slots = pkcs11.getSlotList()
# ユーザー認証用スロットを取得
slot = slots[0]
```

2.6.3 セッションを開いてログイン

続いて暗証番号を入力します。暗証番号が間違っていたら PyKCS11.PyKCS11Error が投げられますのでうまくハンドリングしてください。

```
# セッションを開く
session = pkcs11.openSession(slot)
# 暗証番号を入力
session.login("XXXX")
```

2.6.4 署名処理

続いて、認証用秘密鍵で署名を行います。まず秘密鍵オブジェクトを取得します。実際にカードから秘密鍵を読み出しているわけではなく、あくまでハンドラを取得しています。ここでは「hello」というメッセージに対して署名します。

```
# 秘密鍵オブジェクトを取得
privkey = session.findObjects([(CKA_CLASS, CKO_PRIVATE_KEY)])[0]
# カード内の秘密鍵で署名
signed = session.sign(privkey, "hello")
```

signed に 256byte の署名値が得られました。

最後に、以下のように終了処理を行います。

```
# ログアウト
session.logout()
# セッションを閉じる
session.closeSession()
```

2.7 マイナンバーカードで PDF 署名

JSignPdf[*2]は PDF 署名を行うソフトウェアです。このツールは PKCS#11 ライブラリをサポートしていますので、OpenSC の PKCS#11 ライブラリを利用することで公的個人認証の証明書で署名を行うことができます。

さっそくやってみましょう。環境は Ubuntu 18.04 LTS を想定していますが、JSignPdf は Java アプリケーションですので他の OS でも一部読み替えていただければ動作するはずです。

OpenSC、Java Runtime のインストール

```
$ sudo apt install -y opensc openjdk-8-jre
```

下記 URL から JSignPdf をダウンロードして zip ファイルを展開します。今回は JSignPdf-1.6.3.zip を利用しました。

http://jsignpdf.sourceforge.net/

zip ファイルを展開したら、設定ファイルを 2 つ編集します。

conf/conf.properties を以下のように編集 (コメントを外すだけです):

```
pkcs11config.path=conf/pkcs11.cfg
```

OpenSC の PKCS11 ライブラリをロードするようにします。

conf/pkcs11.cfg を以下のように編集:

```
name=OpenSC-PKCS11
library=/usr/lib/x86_64-linux-gnu/pkcs11/opensc-pkcs11.so
slot=1
```

ここで slot=1 を指定しているのは公的個人認証のデジタル署名用証明書で署名するためです。(slot=0 はユーザー認証用の証明書に対応します)

JSignPdf を実行します。

```
$ java -jar JSignPdf.jar
```

[*2] http://jsignpdf.sourceforge.net/

図 2.2　JSignPdf の UI

JSignPDF の UI ではキーストアタイプに「PKCS11」を選択し、キーストアパスワードに公的個人認証の電子署名用パスワードを入力します。パスワードのアルファベットは大文字で入力してください。入力 PDF ファイルと出力 PDF ファイルを指定して「Sign It」ボタンを押すと署名できます。

これで PDF 署名が完了しましたが、正しく署名できたのか不安になることでしょう。JSignPdf には簡易的な検証を行うツール (Verifier.jar) も付属しておりこれを使って署名済み PDF を検証できます。

```
# JPKI デジタル署名用 CA 証明書を読み出し
$ pkcs15-tool --read-certificate 4 > ca.pem
$ java -jar Verifier.jar signed.pdf -c ca.pem
```

出力結果に `fails=no` という行があれば CA 証明書による署名検証は成功しています。ただこちらのツールはデバッグツールのようで通常これを使うことはないでしょう。

Windows 上の Adobe Acrobat Reader DC で署名済み PDF を開くと次の様に表示されます。[*3]

[*3] Acrobat Reader DC の「環境設定」->「検証」->「Windows 統合」->「署名を検証」にチェックをいれると、Windows の証明書ストアを参照するようになります。

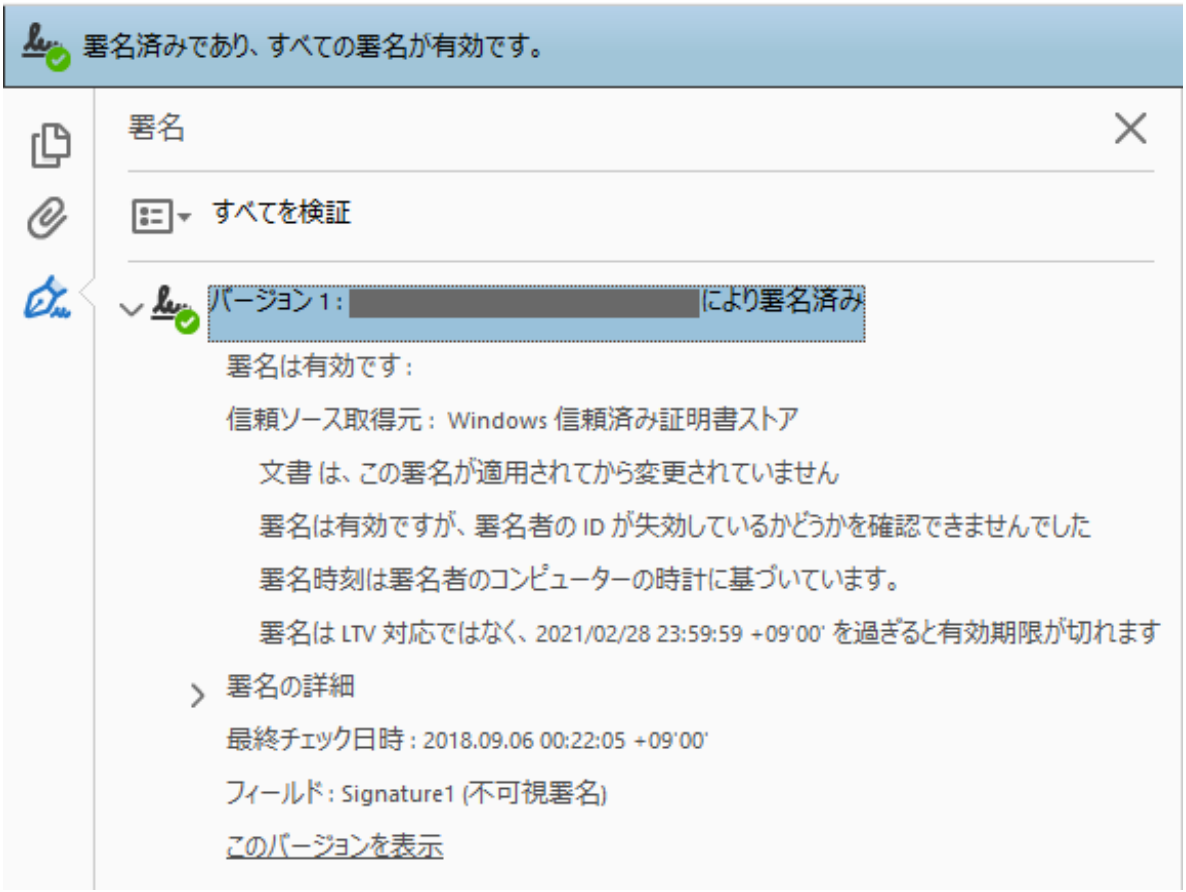


図 2.3　Adobe Acrobat Reader DC の署名パネル

もちろんこれは失効情報などを確認しない簡易的な署名検証です。PDF に署名された証明書から CA 署名書までの信頼のパスを構築できたことを意味します。

2.7.1 まとめ

現在、法務局で行われる登記などの手続きは公的個人認証で署名した PDF ファイルでの電子申請を受け付けているそうです。

実際に今回の方法で署名した PDF ファイルが法務局で受理されるのかどうか、筆者は試していません（誰か試してみてください）。

普段 Linux デスクトップを利用されている皆さんは、様々な行政サービスを受けられず、悲しい思いをされていることでしょう。こんな風にプラットホーム非依存な方法で様々な行政手続きが出来るようになると嬉しいですね。

2.8 マイナンバーカードで SSH する

公的個人認証 AP にはユーザー認証用の証明書と秘密鍵が入っています。この秘密鍵は RSA 2048bit 鍵ですので SSH(Secure Shell) の公開鍵認証に利用できそうです。SSH は安全にリモートホストにログインし、操作する為のプロトコルです。インターネットに携わるエンジニアであれば日常的に SSH を利用されていると思いますので詳細は省きますが、SSH プロトコルについて簡単に説明しておきます。

クライアントとサーバーはまずバージョン情報を交換し、DH 鍵交換、ホスト認証を行います。これにより暗号化されたトランスポート層を確立します。

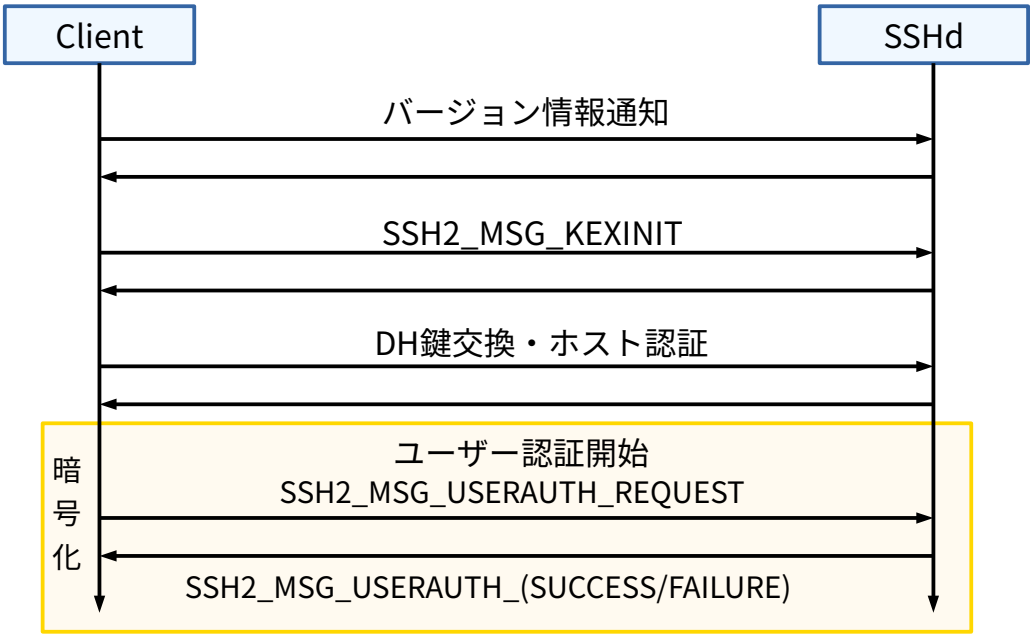


図 2.4　SSH プロトコル

続いて認証方式のネゴシエーションを行い、公開鍵方式、あるいはパスワード認証を行うかどうかを決定します。

SSH の公開鍵認証方式では図のようなデータ構造に対して署名を行います。今回は公的個人認証のユーザー認証用秘密鍵で署名を行い SSH 認証をやってみます。

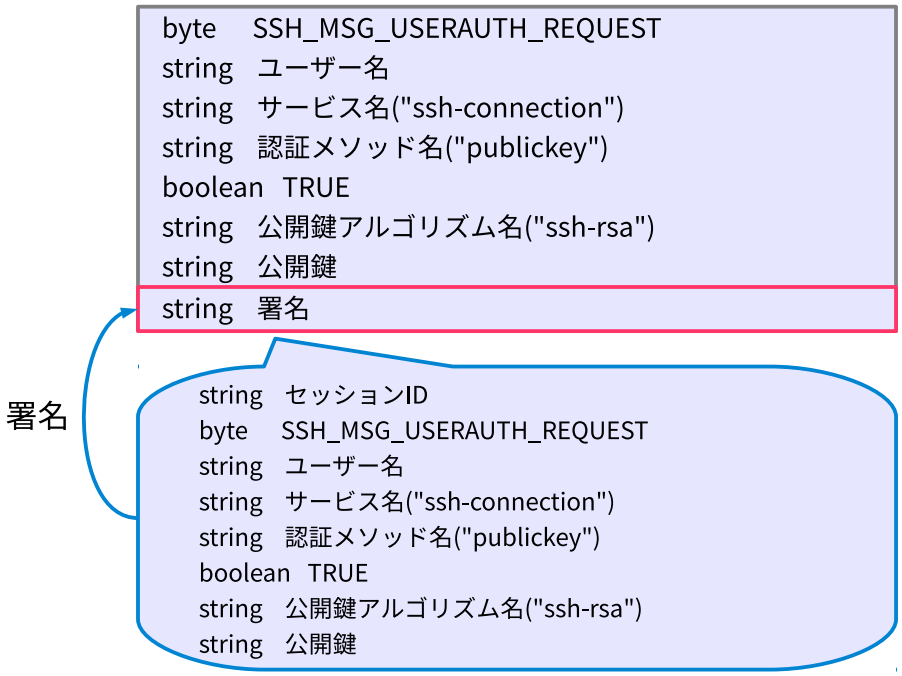


図 2.5　SSH 公開鍵認証方式

まずは認証用公開鍵を OpenSSH 形式で取り出して、authorized_keys に登録しましょう。ユーザー認証用公開鍵は番号 1 に対応します。

```
$ pkcs15-tool --read-ssh-key 1 > id_rsa.pub
```

この公開鍵をサーバー側の authorized_keys に登録したら、いよいよ SSH ログインです。

OpenSSH は PKCS#11 API に対応していますので-I オプションでさきほどビルドした PKCS11 ライブラリを指定します。(~/.ssh/config に PKCS11Provide を指定できるので通常この長いオプションは不要です。)

```
$ ssh -I /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so hostname
Enter PIN for 'JPKI (User Authentication PIN)':
```

このように PIN 入力を求められるので役所で設定した 4 桁数字の暗証番号を入力します。これで図に示したデータに対して署名を行い、認証成功となります。

今回は SSH 認証なので証明書は使いませんでしたが、証明書の失効情報を得るにはなぜか総務大臣の認可が必要だそうなので証明書の検証が必要な場合は面倒ですが申請するしかないですね。

2.9 マイナンバーカードで PAM 認証

Linux をサーバーとして利用している場合、普通は SSH で公開鍵認証を利用するのでパスワード認証を使う機会は減っただろうと思います。しかし、デスクトップ Linux へログインする際のローカル認証はどうでしょうか。まだまだパスワード認証が使われていますし、オンプレで設置している物理サーバーの root パスワードを複数人で共有するといった運用もまだ残っているのではないでしょうか。

こんな時、物理的な認証トークンがあればずっと安全な運用になるのですが、私達はそんなデバイスを既に持っているはずです。そう、マイナンバーカードです。

ここではマイナンバーカード内の鍵で Linux にログインしたり、sudo したり、多要素認証デバイスとしてカードを活用する方法を紹介します。

2.9.1 PAM

Linux には PAM(Pluggable Authentication Module) と呼ばれる認証モジュールとアプリケーションを分離するための仕組みが用意されています。デフォルトの認証モジュール (pam_unix.so) は`/etc/shadow` を参照してパスワード認証を行っていますが、パスワード認証以外にも様々な認証モジュールが用意されていて、独自の認証モジュールを自作することもできます。

PKCS#11 に対応したデバイスで認証するための pam_p11 という PAM モジュールがあります。マイナンバーカードは既に OpenSC 経由で PKCS#11 API を扱えるようになっていますので、pam_p11 から OpenSC の pkcs11 モジュールをロードすればうまくいきそうです。

2.9.2 インストール

```
$ sudo apt install -y opensc libpam-p11
```

2.9.3 公開鍵の設置

pam_p11 はユーザーのホームディレクトリに配置した公開鍵または証明書を参照してカードの所有を確認します。公開鍵は SSH と同様に `~/.ssh/authorized_keys` に設置します。ちょっと紛らわしいですが sshd だけでなく login や sudo などのプログラムが PAM 経由でこのファイルを参照するということです。

以下のコマンドで公的個人認証の公開鍵を配置します。

```
$ mkdir ~/.ssh/
$ pkcs15-tool --read-ssh-key 1 >> ~/.ssh/authorized_keys
```

あるいは、証明書で認証する場合は~/.eid/authorized_certificates に PEM 形式の証明書を配置します。

以下のコマンドで公的個人認証の証明書を配置します。

```
$ mkdir ~/.eid/
$ pkcs15-tool -r 1 >> ~/.eid/authorized_keys
```

2.9.4 PAM の設定

続いて PAM の設定を行いますが、今回はマイナンバーカードを利用した認証方式として、

1. マイナンバーカードだけで認証
2. マイナンバーカードと UNIX パスワードの両方を必要とする二要素認証
3. マイナンバーカードと UNIX パスワードのどちらか一つを必要とする認証

の構成例を紹介します。

Debian や Ubuntu では共通の PAM 設定ファイル/etc/pam.d/common-auth に以下の様な記述が見つかるはずです。

```
auth    [success=1 default=ignore]    pam_unix.so nullok_secure
auth    requisite                     pam_deny.so
(略)
```

この辺りを変更していきます。

2.9.5 マイナンバーカードだけで認証

これがマイナンバーカードだけで認証する設定例です。pam_unix.so の設定を pam_p11_openssh.so で置き換えます。

```
-auth   [success=1 default=ignore]    pam_unix.so nullok_secure
+auth   [success=1 default=ignore]    pam_p11_openssh.so opensc-pkcs11.so
 auth   requisite                     pam_deny.so
(略)
```

pam_p11_openssh.so は設置した公開鍵~/.ssh/authorized_keys を参照します。証

明書を設置した場合は pam_p11_openssh.so を pam_p11_opensc.so に置き換えてください。

Ubuntu のログイン画面 (gdm3) ではこの様にパスワードの代わりにカードの暗証番号を入力します。

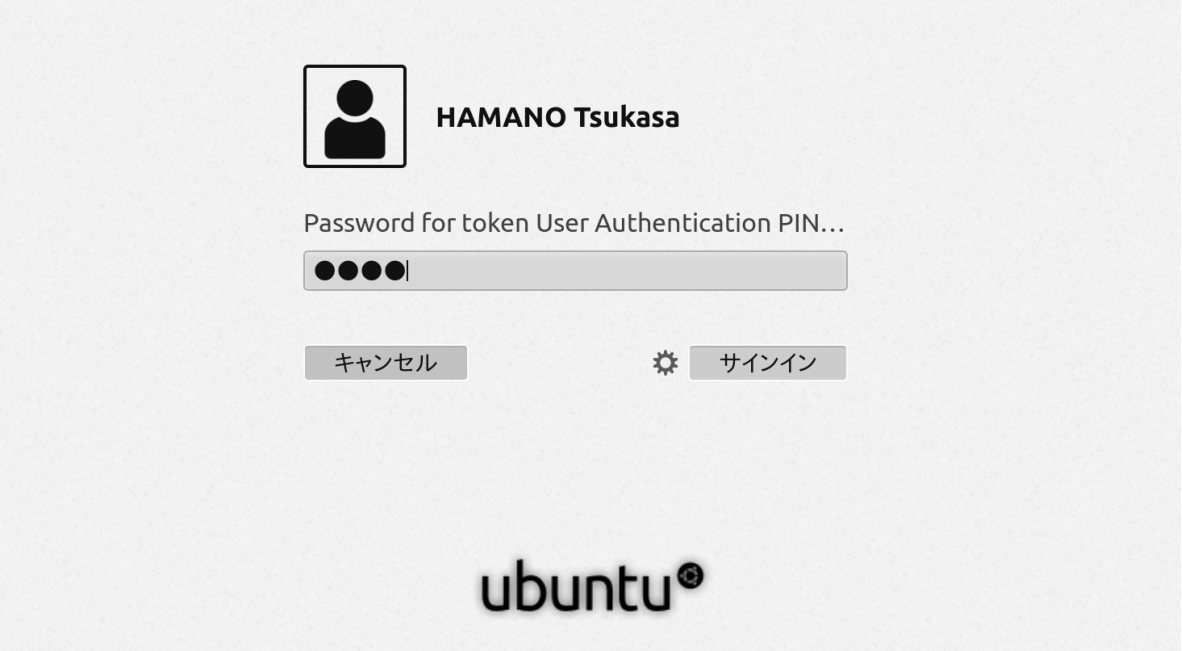


図 2.6　ログイン画面

他のログインマネージャーでも動作するはずです。

2.9.6 マイナンバーカードと UNIX パスワードの両方を必要とする二要素認証

以下のように pam_unix.so の前に pam_p11 の設定を記述します。

```
+auth  [success=1 default=ignore]    pam_p11_openssh.so opensc-pkcs11.so
+auth  requisite                     pam_deny.so
 auth  [success=1 default=ignore]    pam_unix.so nullok_secure
 auth  requisite                     pam_deny.so
(略)
```

たとえばこれで sudo su を実行すると JPKI 認証用の暗証番号を入力した後に通常のパスワードを入力し両方の認証が通ってから root になれます。

```
$ suod su
Password for token User Authentication PIN (JPKI): ****
[sudo] hamano のパスワード: ********
# id
uid=0(root) gid=0(root) groups=0(root)
```

2.9.7 マイナンバーカードと UNIX パスワードのどちらか一つを必要とする認証

```
+auth  [success=2 default=ignore]    pam_p11_openssh.so opensc-pkcs11.so
 auth  [success=1 default=ignore]    pam_unix.so nullok_secure
 auth  requisite                     pam_deny.so
(略)
```

/etc/pam.d/以下の設定は間違えるとログインできなくなるので慎重にやりましょう。

2.10 マイナンバーカードで macOS にログイン

macOS はスマートカードによる認証をサポートしています。スマートカード認証を有効にすると、パスワードの代わりにカードを使って macOS にログインできます。OS へのログインだけでなくスクリーンロックや sudo、ブラウザでの TLS クライアント認証でもカードを利用できます。

ここではマイナンバーカードで自分の macOS にログインし、スクリーンロックや sudo で利用する方法を紹介します。

2.10.1 macOS にログイン

OpenSC が macOS のスマートカード認証に対応したのはバージョン 0.18.0 以降です。以下のページから最新版の OpenSC をダウンロードして dmg ファイル内のインストーラーを実行してインストールしてください。

https://github.com/OpenSC/OpenSC/wiki

まず、ユーザーアカウントとカードに入っているユーザー認証用証明書のペアリングを行います。マイナンバーカードをリーダーに挿入し、sc_auth hash コマンドでユーザー認証用証明書のハッシュ値を確認します。

```
$ sc_auth hash
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX User Authentication Key
(略)
```

そして root 権限で sc_auth accept コマンドを実行し、ユーザーアカウントと証明書を紐付けます。

```
$ sudo sc_auth accept -u <username> -h XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

macOS のログインで利用する証明書は、信頼されたルート認証局からのパスが構築されている必要があります。キーチェーンアクセスで CA 証明書を右クリックして「常に信頼」をやってしまいそうですが、これではログイン中のユーザーコンテキストに限定されてしまうので上手くいきません。CA 証明書はシステムキーチェーンにインポートして信頼する必要があります。

まず pkcs15-tool コマンドを実行し、ユーザー認証用 CA 証明書をマイナンバーカードから取り出します。次の通りハッシュ値が一致すれば間違いなく地方公共団体情報システム機構 (J-LIS) のユーザー認証用 CA 証明書です。

```
$ pkcs15-tool -r 3 > ca.pem
$ openssl x509 -noout -subject -fingerprint -in ca.pem
subject=C = JP, O = JPKI, OU = JPKI for user authentication,
OU = Japan Agency for Local Authority Information Systems
SHA1 Fingerprint=11:D9:CC:90:49:2B:10:FC:FA:D1:DA:D3:08:65:C2:40:98:39:13:D0
```

つづいて、この CA 証明書をシステムのキーチェーンにインポートして信頼します。

```
$ sudo security add-trusted-cert -d -r trustRoot \
  -k /Library/Keychains/System.keychain ca.pem
```

図 2.7 macOS キーチェーン

最後にスマートカードログオンを有効にして設定完了です。

```
$ sudo security authorizationdb smartcard enable
$ security authorizationdb smartcard status
Current smartcard login state: enabled \
(system.login.console enabled, authentication rule enabled)
```

スマートカードログオンを無効化するには以下を実行します。

```
$ sudo security authorizationdb smartcard disable
```

ログイン画面では通常以下のようにユーザー名とパスワードを入力します

図 2.8　macOS パスワード認証

カードを読み込むと...

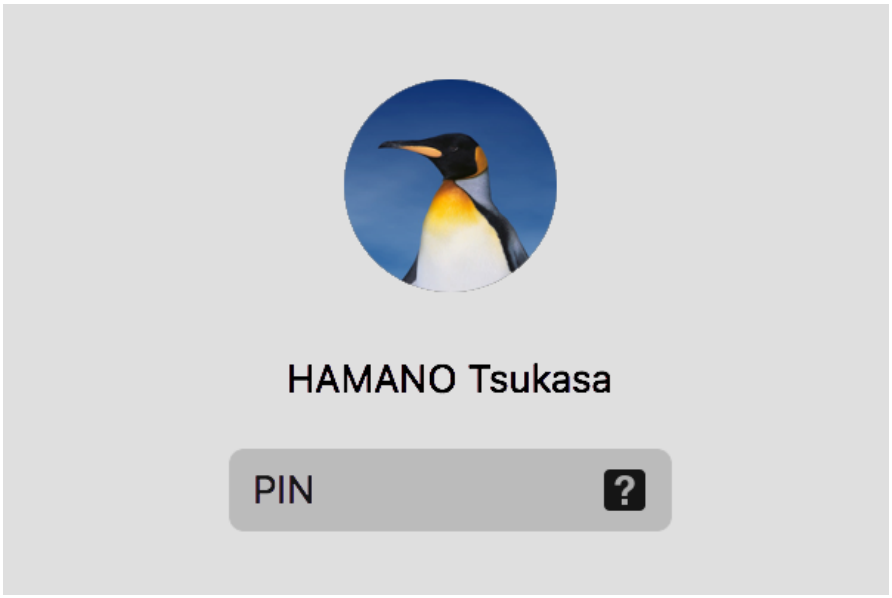


図 2.9　macOS スマートカード認証

この様にログイン画面のパスワード入力フォームが PIN の入力フォームに変わります。ユーザーアカウントと紐付いているため、ユーザーアカウントの選択画面でカードを刺すと自動的にそのユーザーを選択するということも出来ます。

また、スクリーンロックもカードを挿入するとこのような PIN の入力フォームになります。

図 2.10　macOS スクリーンロック

ターミナルで sudo を実行した場合もパスワードでなくスマートカードを利用して root 権限を利用できます。

```
$ sudo su
Enter PIN for 'JPKI':
sh-3.2#
```

2.10.2 課題

- ログイン画面に PIN の失敗回数が表示されないのが不親切。
  - 3 回間違えるとロックされて役所で再設定することになるので要注意。PIN の失敗回数 (あと何回間違えてよいのか) がわからなくなったら、`pkcs15-tool --list-pins` や myna コマンド[*4]で確認できます。
- カードによるログイン後にキーチェーンにログインするパスワードを求められる。
  - OSX はログイン時に入力したパスワード (もしくは PIN) で同時にキーチェーンにもログインするようだ、つまりキーチェーンのパスワードをカードの PIN と同じに設定すれば面倒はなくなるが、これはやりたくない。

[*4] https://github.com/jpki/myna

2.10.3 Apple とスマートカード認証

macOS のスマートカード認証の歴史は 2005 年の米国大統領令 HSPD-12 に遡ります。これは米国連邦職員に安全で信頼性のある認証を義務付けるというものでした。NIST(アメリカ国立標準技術研究所) はこの HSPD-12 に応じて FIPS 201(PIV カード仕様) の標準化を行い、Apple はこれに準拠するための CDSA/Tokend フレームワークを OS X 10.4 から提供し始めました。

こうして macOS で PIV カード (米国公務員が利用) や CAC カード (米軍が利用) の利用環境が整備されていきました。しかし、この CDSA フレームワークは OS X 10.7 Lion 以降で非推奨となります。そして、PIV や CAC のための tokend が標準で付属しなくなりオープンソースコミュニティによるサポートに切り替わりました。[*5]

Apple は今後利用するセキュリティフレームワークとして CryptoTokenKit を推奨していますが、詳細なドキュメントが無いために PIV や CAC の対応が進んでいません。

Apple からもっと情報が出てくれば、OpenSC プロジェクトも CryptoTokenKit に移行するという動きも出てくると思われますが現状まだ macOS High Sierra で CDSA フレームワークを利用できるので、まだこのままでいいやという状況です。

いきなり CDSA フレームワークが無くなると困る人が多いのでまだしばらくは無くならないだろうと思われます。

[*5] https://lists.apple.com/archives/fed-talk/2011/Jul/msg00099.html

あとがき

まだ多くの人が活用できていないであろうマイナンバーカードの活用例を紹介しました。広く普及するにはまだ時間がかかりそうですが、マイナンバーカードは紙、印鑑、パスワード認証などあらゆるレガシーを終わらせる可能性を秘めた素敵デバイスです。

様々な公共システムが IE 限定で Linux デスクトップで使えなくて悲しいという事を書きましたが、一般論として対応プラットフォームの拡大は悩ましい問題です。この様な公共システムには多大な税金が使われているからです。お金が無いから仕方がないね、と諦めてしまいそうですが電子先進国と言われるエストニアに目を向けてみると、エストニアの国民 ID カードシステムは Windows,macOS,Linux と幅広く対応している事がわかります。エストニアの人口は日本の 100 分の 1、新しい国なので過去のしがらみがないなど、日本の行政システムと多くの条件が異なるので直接比較することは難しいでしょう。

しかしシステム開発に利用できる予算が日本と比べて少ないエストニアが Linux デスクトップをサポート出来るのはなぜでしょうか。ID カードの関連ソフトウェアを github.com などのコミュニティでメンテナンスすることで少コストを実現しているようです。日本もこの様なオープンな姿勢を参考にしてみても良いのではないでしょうか。

参考文献

- 『個人番号カードプロテクションプロファイル 第 1.00 版』
  - https://www.ipa.go.jp/security/jisec/certified_pps/c0431/c0431_it4485.html
- 『IC・ID カードの相互運用可能性の向上に係る基礎調査』
  - https://www.ipa.go.jp/files/000024558.pdf
- 小松文子 (他著),『改訂　PKI ハンドブック』, ソフト・リサーチ・センター
- ラウル アリキヴィ (著)・前田陽二 (著),『未来型国家エストニアの挑戦　電子政府がひらく世界』, インプレス R&D
- 手塚 悟 (著), マイナンバーで広がる電子署名・認証サービス, 日経 BP

マイナンバーカードと電子署名の本

2018 年 10 月 8 日　第 1 版発行 (技術書典 5)
2019 年 9 月 22 日　第 2 版発行 (技術書典 7)

著者　濱野　司
Twitter　@hamano
表紙　Ai
印刷　日光企画